### SOFTWARE MAINTENANCE MANUAL

FOR THE

AIRNET AEROMODEL AND

WEAPONS MODEL CONVERSION

VOLUME 1 of 3

Rev. 0.0: 31 March 1993

CONTRACT NO. N61339-91-D-0001

D.O.: 0014

CDRL SEQUENCE NO. A005

Prepared for:

## **STRICOM**

Simulator Training and Instrumentation Command

Simulator Training and Instrumentation Command
Naval Training Systems Center
12350 Research Parkway
Orlando, FL 32826-3275

Prepared by:

### LDRAL

ADST Program Office 12443 Research Parkway, Suite 303 Orlando, FL 32826

DIC QUALITY INEPECTED 5

19941128 014

DISTRIBUTION STATEMENT A

Approved for public release;
Distribution Unlimited

# REPORT DOCUMENTATION PAGE

Form approved OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. send comments regarding this burden estimate or any other aspect of talks collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget Project (0704-0188), Washington, DC 20503. 3. REPORT TYPE AND DATES COVERED 2. REPORT DATE 1. AGENCY USE ONLY (Leave blank) Version 0.0 31 March 1993 5. FUNDING NUMBERS 4. TITLE AND SUBTITLE Advanced Distributed Simulation Technology Software Maintenance Manual Contract No. N61339-91-D-0001 for the AIRNET Aeromodel and Weapons Model Conversion; Volume 1 of 3 6. AUTHOR(S) Accesion For Branson, Roger; McCarter, Steve NTIS CRA&I DTIC TAB Unannounced 8, PERFORMING ORGANIZATION 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Justification \_\_\_\_ REPORT NUMBER Loral Systems Company ADST/WDL/TR-93-W003092 ADST Program Office CDRL A005 12443 Research Parkway, Suite 303 Orlando, FL 32826 Distribution / Availability Codes 10. SPONSORING 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) ORGANIZATION REPORT Simulation, Training and Instrumentation Command Avail and or Dist Special **STRICOM** Naval Training Systems Center 12350 Research Parkway Orlando, FL 32826-3275 11. SUPPLEMENTARY NOTES 12b. DISTRIBUTION CODE 12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited. 13. ABSTRACT (Maximum 200 words) The ADST Software Maintenance Manual provides guidance for modifying the aeromodel and weapons model data files. 15. NUMBER OF PAGES 14. SUBJECT TERMS 689 16, PRICE CODE

17. SECURITY CLASSIFICATION

OF ABSTRACT

UNCLASSIFIED

17. SECURITY CLASSIFICATION

OF THIS PAGE

**UNCLASSIFIED** 

OF REPORT

17. SECURITY CLASSIFICATION

UL

20. LIMITATION OF ABSTRACT

1.	Scope	1
1.1.	Identification	
1.2.	System overview	l
1.3.	Document overview	1
2.	Referenced documents	2
3.	Modifiable data	3
3.1	Aero_data	4
3.1.1	MOMENT_OF_INERTIA_X	4
3.1.2	MOMENT_OF_INERTIA_Y	5
3.1.3	MOMENT_OF_INERTIA_Z	6
3.1.4	AIRFRAME_MASS	6
3.1.5	ORDINANCE MASS	7
3.1.6	GRAV_CONSTANT	8
3.1.7	CG AC X	9
3.1.8	CG AC Y	10
3.1.9	CG AC Z	10
3.1.10	VIRTUAL WING AREA	11
3,1.11	VIRTUAL_WING_COP_AC_X	12
3.1.12	VIRTUAL WING COP_AC_Y	13
3.1.13	VIRTUAL WING_COP_AC_Z	13
3.1.14	WING LIFT COEFFICIENT FIT 3	14
3.1.15	WING LIFT COEFFICIENT_FIT_2	15
3.1.16	WING LIFT COEFFICIENT_FIT_1	16
3.1.17	WING LIFT COEFFICIENT FIT 0	17
3.1.18	WING STALL AOA	18
3.1.19	VSTAB AREA	19
3.1.20	VSTAB COP AC X	19
3.1.21	VSTAB COP AC Y	20
3.1.22	VSTAB COP AC Z	21
3.1.23	VSTAB LIFT COEFFICIENT 1	22
3.1.24	VSTAB STALL SSA	23
3.1.25	MAIN ROTOR COP AC X	24
3.1.26	MAIN ROTOR COP AC Y	25
3.1.27	MAIN ROTOR COP AC Z	25
3.1.28	MAIN ROTOR MAX THRUST	26
3.1.29	MAIN ROTOR MAST TILT	27
3.1.30	MAIN ROTOR MAX LOAD TORQUE	28
3.1.31	MAIN ROTOR MAX PITCH MOMENT	29
3.1.32	MAIN ROTOR MAX ROLL MOMENT	30
3.1.33	MAIN ROTOR TOROUE COUPLING GAIN	
3.1.34	MAIN ROTOR GROUND EFFECT FACTOR	32
3.1.35	TAIL ROTOR COP AC X	33
3.1.36	TAIL ROTOR COP AC Y	34
3.1.37	TAIL ROTOR COP AC Z	35
3.1.38	TAIL ROTOR MAX THRUST	36

# Reference # W003092; 31 March 1993 VOLUME 1 of 3; Rev. 0.0

3.1.39	TAIL_ROTOR_MAX_LOAD_TORQUE	37
3.1.40	P_DRAG_COEFF_CONST	38
3.1.41	P_DRAG_TAS_BREAK	40
3.1.42	P_DRAG_COEFF_BREAK	41
3.1.43	P DRAG TAS MAX	42
3.1.44	P DRAG COEFF MAX	42
3.1.45	TOTAL WETTED_SURFACE_AREA	43
3.1.46	MAX_ATT_CTL_ANGLE_STOP	43
3.1.47	MAX_ATT_DAMPING_FACTOR	44
3.1.48	HOVER SLOW LIMIT	45
3.1.49	HOVER_AUG_PITCH_RESET_VALUE	47
3.1.50	MAX_ATT_CTL_ANGLE_NORM	48
3.1.51	MAX_ATT_CTL_ANGLE_MED	
3.1.52	MAX_ATT_CTL_ANGLE_SLOW	51
3.1.53	HOVER MED LIMIT	52
3.1.54	ATT_CTL_PITCH_P_GAIN	54
3.1.55	ATT_CTL_PITCH_I_GAIN	54
3.1.56	ATT_CTL_ROLL_P_GAIN	55
3.1.57	ATT_CTL_ROLL_I_GAIN	57
3.1.58	HOVER_AUG_ROLL_P_GAIN	58
3.1.59	HOVER_AUG_ROLL_I_GAIN	
3.1.60	HOVER_AUG_PITCH_P_GAIN	61
3.1.61	HOVER_AUG_PITCH_I_GAIN	62
3.1.62	HOVER_AUG_YAW_P_GAIN	64
3.1.63	HOVER_AUG_YAW_I_GAIN	65
3.1.64	HOVER_AUG_CLIMB_P_GAIN	66
3.1.65	HOVER_AUG_CLIMB_I_GAIN	68
3.1.66	MAX_STAB_AUG_PITCH_ROLL_CONTROL	69
3.1.67	MAX_STAB_AUG_YAW_CLIMB_CONTROL	71
3.1.68	ROLL_RATE_DAMPING_GAIN	72
3.1.69	PITCH_RATE_DAMPING_GAIN	74 
3.1.70	YAW_RATE_DAMPING_GAIN	75
3.1.71	VERTICAL_RATE_DAMPING_GAIN	
3.1.72	LATERAL_VELOCITY_DAMPING_GAIN	77
3.1.73	LIFT_COEFF_VIRTUAL_WING	78
3.1.74	OSWALD_EFFIC_FACTOR	79 
3.1.75	INDUCED_DRAG_COEFF	79
3.2	Aero_init	80
3.2.1	Cyclic_pitch	80
3.2.2	Cyclic_roll	83
3.2.3	Collective	85
3.2.4	Pedal	88
3.2.5	Stab_aug_pitch_integrator	89
3.2.6	Stab aug roll integrator	90
3.2.7	Stab aug yaw integrator	90
3.2.8	Stab aug climb integrator	93

3.2.9	Attitude_control_pitch_integrator	94
3.2.10	Attitude_control_roll_integrator	95
3.2.11	Hover_aug_pitch_integrator	96
3.2.12	Hover_aug_roll_integrator	97
3.2.13	Hover_aug_pitch_angle	99
3.2.14	Hover_aug_roll_angle	100
3.3	Aero_simple	102
3.3.1	MAX_HELICOPTER_POWER	102
3.3.2	MAX_HH	103
3.3.3	H_K1	104
3.3.4	H_K2	105
3.3.5	H_K7	105
3.3.6	H_K8	106
3.3.7	H_KP	107
3.3.8	H_KPR	107
3.3.9	H_KY	
3.3.10	H_KH	109
3.3.11	H_CHH	110
3.3.12	H_CL	111
3.4	Aero_stealth	
3.4.1	H_FWD_MUL	112
3.4.2	H_SIDE_MUL	113
3.4.3	H_COLL_MUL	113
3.4.4	MAX_TORQUE	114
3.4.5	MAX_FORCE	
3.4.6	MASS	
3.4.7	INERTIA	
3.4.8	DEAD_ZONE	117
3.5	Engine_data	118
3.5.1	GOVERNOR_ENGINE_SPEED_SETTING	118
3.5.2	GOVERNOR_P_GAIN	119
3.5.3	GOVERNOR_I_GAIN	120
3.5.4	MAX_ENGINE_TORQUE	121
3.5.5	MIN_ENGINE_LOAD_TORQUE	122
3.5.6	MAX_ENGINE_PERCENT_POWER	123
3.5.7	ENGINE_TORQUE_INTERCEPT	124
3.5.8	ENGINE_TORQUE_SLOPE	124
3.5.9	NOSE_GEARBOX_RATIO	125
3.5.10	MAIN_ROTOR_GEAR_RATIO	126
3.5.11	TAIL_ROTOR_GEAR_RATIO	127
3.5.12	POWERTRAIN_INERTIA	128
3.5.13	MAX FUELFLOW	128
3.6	Engine_init_data	129
3.6.1	Engine power	129
3.6.2	Engine percent torque	131
363	Fngine speed	132

3.6.4	Integrator_gain	135
3.6.5	Last percent shaft speed	136
3.6.6	Last percent torque	13/
3.6.7	Hours of flight	136
3.7	Fnoine stat data	139
3.7.1	Minutes of flight	139
3.7.2	Old minutes of flight	140
3.7.3	Engine status	
3.7.4	Starting engine	143
3.7.5	Number of engines	144
3.7.6	Engine is damaged	145
3.7.7	Transmission is damaged	14/
3.8	Kinemat_data	148
3.8.1	GRAV CONSTANT	148
3.8.2	SIN_AOA_LIMIT	149
3.8.3	COS AOA LIMIT	149
3.8.4	SIN_YAW_LIMIT	150
3.8.5	COS YAW LIMIT	150
3.8.6	DISPLAY SPEED LIMIT	151
3.9	Kinemat init data	152
3.9.1	Pos unit vel	152
3.9.2	Neg unit vel	153
3.9.3	Sin ana	104
3.9.4	Cos aga	133
3.9.5	Sin vaw	157
3.9.6	Cos_yaw	140
3.9.7	Altitude	100
3.9.8	Body_pitch	101
3.9.9	Body_pitch_offset	162
3.9.10	Velocity_pitch	163
3.9.11	Roll	103 164
3.9.12	Heading	165
3.9.13	True_airspeed	168
3.9.14	Indicated_airspeed	168
3.9.15	G_force	169
3.9.16	Vertical_speed	170
3.9.17	Gravity	
3.9.18	Norm_vel	172
3.10	Hellfr_miss_char	179 179
3.10.1	HELLFIRE_ARM_TIME	
3.10.2	HELLFIRE_BURNOUT_TIME	
3.10.3	HELLFIRE_MAX_FLIGHT_TIME	170
3.10.4	SPEED 0	
3.10.5	THETA ()	
3.10.6	SIN_UNGUIDE	
3 10 7	COS UNGUIDE	

3.10.8	SIN_CLIMB	
3.10.9	COS_CLIMB	181
3.10.10	SIN_LOCK	182
3.10.11	COS_LOCK	183
3.10.12	COS_TERM	184
3.10.13	COS_LOSE	185
3.11	Hellfr_miss_poly_deg	
3.11.1	HELLFIRE_TOF_DEG	
3.11.2	HELLFIRE BURN SPEED DEG	
3.11.3	HELLFIRE_COAST_SPEED_DEG	
3.12	Hellfire tof coeff	
3.13	Hellfire_burn_speed_coeff	
3.14	Hellfire_coast_speed_coeff	
3.15	Maverick_miss_char	
3.15.1	MAVERICK ARM TIME	
3.15.2	MAVERICK_BURNOUT_TIME	194
3.15.3	MAVERICK MAX FLIGHT TIME	
3.15.4	MAVERICK_LOCK_THRESHOLD	196
3.15.5	MAVERICK_HOLD_THRESHOLD	
3.15.6	SPEED_0	
3.15.7	THETA_0	
3.15.8	SIN UNGUIDE	201
3.15.9	COS UNGUIDE	203
3.15.10	SIN_CLIMB	204
3.15.11	COS_CLIMB	206
3.15.12	SIN_LOCK	207
3.15.13	COS_LOCK	209
3.15.14	COS_TERM	210
3.15.15	COS_LOSE	212
3.16	Maverick_miss_poly_deg	213
3.16.1	MAVERICK_BURN_SPEED_DEG	
3.16.2	MAVERICK_COAST_SPEED_DEG	
3.17	Maverick_burn_speed_coeff	216
3.18	Maverick_coast_speed_coeff	218
3.19	Stinger_miss_char	219
3.19.1	STINGER_BURNOUT_TIME	219
3.19.2	STINGER_MAX_FLIGHT_TIME	221
3.19.3	STINGER_LOCK_THRESHOLD	222
3.19.4	SPEED_0	
3.19.5	THETA_0	224
3.19.6	INVEST_DIST_SQ	225
3.19.7	FUZE_DIST_SQ	
3.20	Stinger_miss_poly_deg	
3.20.1	STINGER_BURN_SPEED_DEG	226
3.20.2	STINGER_COAST_SPEED_DEG	
3.21	Stinger burn speed coeff	230

# Reference # W003092; 31 March 1993 VOLUME 1 of 3; Rev. 0.0

3.22	Stinger_coast_speed_coeff	231
3.23	Tow_miss_char	233
3.23.1	TOW_BURNOUT_TIME	
3.23.2	TOW_RANGE_LIMIT_TIME	234
3.23.3	TOW_MAX_FLIGHT_TIME	235
3.24	Tow_miss_poly_deg	236
3.24.1	TOW_BURN_SPEED_DEG	
3.24.2	TOW_COAST_SPEED_DEG	238
3.24.3	TOW_BURN_TURN_DEG	239
3.24.4	TOW_COAST_TURN_DEG	241
3.25	Tow_burn_speed_coeff	243
3.26	Tow_coast_speed_coeff	
3.27	Tow_burn_turn_coeff	246
3.28	Tow_coast_turn_coeff	
3.29	Adat miss char	250
3.29.1	ADAT_BURNOUT_TIME	250
3.29.2	ADAT_MAX_FLIGHT_TIME	251
3.29.3	INVEST_DIST_SQ	252
3.29.4	HELO_FUZE_DIST_SQ	253
3.29.5	AIR_FUZE_DIST_SQ	
3.29.6	ADAT_TEMP_BIAS_TIME	256
3.29.7	CLOSE_RANGE	257
3.30	Adat_miss_poly_deg	259
3.30.1	ADAT_BURN_SPEED_DEG	259
3.30.2	ADAT_COAST_SPEED_DEG	261
3.30.3	ADAT_BURN_TURN_DEG	262
3.30.4	ADAT_COAST_TURN_DEG	263
3.30.5	ADAT_TEMP_BIAS_DEG	265
3.31	Adat burn speed coeff	266
3.32	Adat_coast_speed_coeff	269
3.33	Adat_burn_turn_coeff	270
3.34	Adat coast turn coeff	272
3.35	Adat_temp_bias_coeff	274
3.36	Atgm_miss_char	276
3.36.1	TOW_BURNOUT_TIME [for ATGM]	276
3.36.2	TOW_RANGE_LIMIT_TIME [for ATGM]	278
3.36.3	TOW_MAX_FLIGHT_TIME [for ATGM]	278
3.36.4	ATGM_TURN_FACTOR	279
3.37	Atgm_miss_poly_deg	280
3.37.1	TOW_BURN_SPEED_DEG [for ATGM]	280
3.37.2	TOW_COAST_SPEED_DEG [for ATGM]	282
3.37.3	TOW BURN TURN DEG [for ATGM]	283
3.37.4	TOW COAST TURN DEG [for ATGM]	285
3.38	Tow burn speed coeff [for ATGM]	287
3.39	Tow_coast_speed_coeff [for ATGM]	289
3.40	Tow burn turn coeff [for ATGM]	29

3.41	Tow_coast _turn_coeff [for ATGM]	<b>2</b> 93
3.42	Kem_miss_char	296
3.42.1	KEM_BURNOUT_TIME	296
3.42.2	KEM_MAX_FLIGHT_TIME	297
3.42.3	KEM_TO_MACH5_FACTOR	298
3.43	Kem_miss_poly_deg	300
3.43.1	KEM_BURN_SPEED_DEG	
3.43.2	KEM_COAST_SPEED_DEG	302
3.43.3	KEM_BURN_TURN_DEG	303
3.43.4	KEM_COAST_TURN_DEG	304
3.44	Kem_burn_speed_coeff	306
3.45	Kem_coast_speed_coeff	308
3.46	Kem_burn_turn_coeff	310
3.47	Kem_coast_turn_coeff	312
3.48	Nlos_miss_char	314
3.48.1	NLOS_LOCK_THRESHOLD	314
3.48.2	NLOS_MAX_TURN_ANGLE	316
3.48.3	NLOS_VERTICAL_FLIGHT_TIME	316
3.48.4	NLOS_DECLINE_FLIGHT_TIME	318
3.48.5	NLOS_LEVEL_FLIGHT_TIME	320
3.48.6	NLOS_ARM_TIME	322
3.48.7	NLOS_BURNOUT_TIME	323
3.48.8	NLOS_MAX_FLIGHT_TIME	323
3.48.9	SPEED_0	324
3.48.10	SPEED_1	325
3.48.11	THETA_0	
3.48.12	SIN_UNGUIDE	
3.48.13	COS_UNGUIDE	327
3.48.14	SIN_CLIMB	327
3.48.15	COS_CLIMB	328
3.48.16	SIN_LOCK	328
3.48.17	COS_LOCK	329
3.48.18	COS_TERM	330
3.48.19	COS_LOSE	330
3.49	Nlos_miss_poly_deg	331
3.49.1	NLOS_BURN_SPEED_DEG	331
3.49.2	NLOS_COAST_SPEED_DEG	332
3.50	Nlos burn speed coeff	332
3.51	Nlos_coast_speed_coeff	333
3.52	Hydra rkt char	334
3.52.1	HYDRA LAUNCHER_POS_X	334
3.52.2	HYDRA LAUNCHER POS Y	336
3.52.3	HYDRA LAUNCHER POS Z	337
3.52.4	SOVIET ARTICULATION	338
3.52.5	HULL_NEG_5_PITCH	339
3.52.6	ARTICULATION MAX	340

3.52.7	ARTICULATION_MIN	341
3.53	Rkt hydra char	341
3.53.1	M151_BURST_SPREAD	342
3.53.2	M261_BURST_HEIGHT	343
3.53.3	M261_BURST_RANGE	345
3.53.4	M261_BURST_SPREAD	346
3.53.5	M255_BURST_RANGE	348
3.53.6	M255_BURST_SPREAD	349
3.53.7	FLECH_60_MAX_RANGE	
3.53.8	HYDRA_MIN_RANGE	351
3.53.9	HYDRA_MAX_RANGE_S5	353
3.53.10	HYDRA_MAX_RANGE_M151	
3.53.11	HYDRA_MAX_RANGE_M261	355
3.53.12	HYDRA_MAX_RANGE_M255	357
3.54	Sub_m73_char	359
3.54.1	Sub m73 char[ 0]	359
3.54.2	M73_FOOT_ANGLE_X	361
3.54.3	M73_FOOT_ANGLE_Y	362
3.55	Sub_flech_char	363
3.55.1	INVEST_DIST_SQ	363
3.55.2	FUZE_DIST_SQ	365
3.55.3	FLECH_60_MAX_RANGE	366
3.56	Flechette_speed_coef	367
4.	Error messages.	369
5.	CSCI data	370
5.1.	Data elements internal to the CSCI	370
6.	Notes	422
Appendix A	- RWA AirNet Call Tree Structure	A-1
Appendix B	- Source code listing for rwa_aerodyn.c	B-1
Appendix C	- Source code listing for rwa engine.c	C-1
Appendix [	)- Source code listing for rwa kinemat.c	D-1
Appendix E	- Source code listing for miss adat.c	£-1
Appendix F	'- Source code listing for miss atgm.c	r-1
Appendix C	G - Source code listing for miss hellfr.c	۲-ی
Appendix F	I - Source code listing for miss_kem.c	H-1
Appendix I	- Source code listing for miss_maverck.c	I-1
Appendix I	- Source code listing for miss_nlos.c	J-1
Appendix K	C - Source code listing for miss_stinger.c	K-1
Appendix I	- Source code listing for miss_tow.c	L-1
Appendix N	1 - Source code listing for rkt hydra.c	M-J
Appendix N	N - Source code listing for rwa_hydra.c	N-1
Appendix (	) - Source code listing for sub_flech.c.	O-1
Appendix F	? - Source code listing for sub_m73.c	P-1
- Trondin I		

## LIST OF TABLES

TABLE 5.1 SUMMARY of DATA ARRAYS	371
TABLE 5.1.1 AERODYNAMICS DATA ARRAY	
TABLE 5.1.2 AERODYNAMICS INITIALIZATION DATA ARRAY	
TABLE 5.1.3 AERODYNAMICS SIMPLE DATA ARRAY	
TABLE 5.1.4 AERODYNAMICS STEALTH DATA ARRAY	
TABLE 5.1.5 ENGINE DATA ARRAY	
TABLE 5.1.6 ENGINE INITIALIZATION DATA ARRAY	
TABLE 5.1.7 ENGINE STATUS DATA ARRAY	
TABLE 5.1.8 KINEMATICS DATA ARRAY	
TABLE 5.1.9 KINEMATICS INITIALIZATION DATA ARRAY	
TABLE 5.1.10 - HELLFIRE MISSILE CHARACTERISTICS DATA ARRAY	
TABLE 5.1.11 HELLFIRE MISSILE POLYNOMIAL DEGREE DATA ARRA	ΑY
	389
TABLE 5.1.12 HELLFIRE MISSILE TIME-OF-FLIGHT DATA ARRAY	389
TABLE 5.1.13 HELLFIRE MISSILE BURN SPEED DATA ARRAY	390
TABLE 5.1.14 HELLFIRE MISSILE COAST SPEED DATA ARRAY	391
TABLE 5.1.15 MAVERICK MISSILE CHARACTERISTICS DATA ARRA	Y
,	392
TABLE 5.1.16 MAVERICK MISSILE POLYNOMIAL DEGREE DATA	
ARRAY	393
TABLE 5.1.17 MAVERICK MISSILE BURN SPEED DATA ARRAY	393
TABLE 5.1.18 MAVERICK MISSILE COAST SPEED DATA ARRAY	
TABLE 5.1.19 STINGER MISSILE CHARACTERISTICS DATA ARRAY	
TABLE 5.1.20 STINGER MISSILE POLYNOMIAL DEGREE DATA ARRA	λΥ
	396
TABLE 5.1.21 STINGER MISSILE BURN SPEED DATA ARRAY	396
TABLE 5.1.22 STINGER MISSILE COAST SPEED DATA ARRAY	396
TABLE 5.1.23 TOW MISSILE CHARACTERISTICS DATA ARRAY	397
TABLE 5.1.24 TOW MISSILE POLYNOMIAL DEGREE DATA ARRAY	397
TABLE 5.1.25 TOW MISSILE BURN SPEED DATA ARRAY	300 200
TABLE 5.1.26 TOW MISSILE COAST SPEED DATA ARRAY	۵۲ <i>۵</i>
TABLE 5.1.27 TOW MISSILE BURN TURN DATA STRUCTURE	399
TABLE 5.1.28 TOW MISSILE COAST TURN DATA STRUCTURE	
TABLE 5.1.29 ADAT MISSILE CHARACTERISTICS DATA ARRAY	
TABLE 5.1.30 ADAT MISSILE POLYNOMIAL DEGREE DATA ARRAY.	402
TABLE 5.1.31 ADAT MISSILE BURN SPEED DATA ARRAY	402
TABLE 5.1.32 ADAT MISSILE COAST SPEED DATA ARRAY	404
TABLE 5.1.33 ADAT MISSILE BURN TURN DATA ARRAY	405
TABLE 5.1.34 ADAT MISSILE COAST TURN DATA ARRAY	406
TABLE 5.1.35 ADAT MISSILE TEMPORAL BIAS DATA ARRAY	00± ⊒∩∖
TABLE 5.1.36 ATGM MISSILE CHARACTERISTICS DATA ARRAY	<del>ኒ</del> ሀ/
TABLE 5.1.37 ATGM MISSILE POLYNOMIAL DEGREE DATA ARRAY	10 <del>1</del> 07
TABLE 5.1.38 ATGM MISSILE BURN SPEED DATA ARRAY	20 <b>:2</b> .
TABLE 5.1.39 ATGM MISSILE COAST SPEED DATA ARRAY	4∪č

## Reference # W003092; 31 March 1993 VOLUME 1 of 3; Rev. 0.0

TABLE 5.1.40 ATGM MISSILE BURN TURN DATA STRUCTURE	409
TABLE 5.1.41 ATGM MISSILE COAST TURN DATA STRUCTURE	410
TABLE 5.1.41 ATGM MISSILE COAST TORN DATA STRUCTURE	111
TABLE 5.1.42 KEM MISSILE CHARACTERISTICS DATA ARRAY	411
TABLE 5.1.43 KEM MISSILE POLYNOMIAL DEGREE DATA ARRAY	411
TABLE J.1.44 REM MIDDIEL BOILT OF LEB DITTI III.	.412
TABLE 5.1.45 KEM MISSILE COAST SPEED DATA ARRAY	413
TABLE 5.1.46 KEM MISSILE BURN TURN DATA ARRAY	.414
TABLE 5.1.47 KEM MISSILE COAST TURN DATA ARRAY	.415
TABLE 5.1.48 NLOS MISSILE CHARACTERISTICS DATA ARRAY	.416
TABLE 5.1.49 NLOS MISSILE POLYNOMIAL DEGREE DATA ARRAY	
TABLE 5.1.50 NLOS MISSILE BURN SPEED DATA ARRAY	
TABLE 5.1.51 NLOS MISSILE COAST SPEED DATA ARRAY	
TABLE 5.1.52 HYDRA ROCKET CONFIGURATION DATA ARRAY	
TABLE 5.1.53 HYDRA ROCKET CHARACTERISTICS DATA ARRAY	
TABLE 5.1.54 SUBMUNITIONS M73 CHARACTERISTICS DATA ARRA	·Υ
	.420
TABLE 5.1.55 SUBMUNITIONS FLECHETTE CHARACTERISTICS DATA	A
ARRAY	.421
TABLE 5.1.56 FLECHETTE SPEED DATA ARRAY	.421

### 1. Scope.

This section shall be divided into the following paragraphs.

#### 1.1. Identification.

This Software Maintenance Manual (SMM) applies to document number WDL/TR92-003011 entitled System Specification for the Rotary Wing Aircraft AirNet Aeromodel and Weapons Model Conversion. This SMM also applies to the AirNet CSCI.

### 1.2. System overview.

The Rotary Wing Aircraft (RWA) system and SIMNET Computer System Configuration Item (CSCI) simulates a manned flight vehicle and associated weapons systems for conducting simulated missions within a controlled database and tactical environment.

#### 1.3. Document overview.

The following paragraphs and subparagraphs identify the data variables, Computer Software Units (CSU) and algorithms that use the data read from data files during initialization. These data files were constructed as a task of the Rotary Wing Aircraft AirNET Aeromodel and Weapons Model Conversion Delivery Order and are documented in detail in Software Design Document for the AirNET Aeromodel and Weapons Model Conversion. Certain CSUs were modified to allow the reading of data values from data files. Computer Software Components (CSC) and CSUs existing in original code are only documented herein to the extent that the data from these data files is used. The original function and operation of the software was not modified. This additional capability allows for the change of variables without requiring a recompile. This SMM is compiled as a guide to the software programmer to assist in understanding how the data variables are used and how a modification of the data will effect computation of certain performance characteristics and limits of the aeromodel, engines, and weapon systems. The modifications to the MCC and communications software are covered in a separate volume.

### 2. Referenced documents.

The following documents are referenced within this document.

WDL/TR--92-003011

SYSTEM SPECIFICATION FOR THE ROTARY WING AIRCRAFT AIRNET AEROMODEL AND WEAPONS MODEL CONVERSION, 6 JUNE 1992.

WDL/TR--93-003036

SOFTWARE DESIGN DOCUMENT FOR THE ROTARY WING AIRCRAFT AIRNET AEROMODEL AND WEAPONS MODEL CONVERSION, 22 JANUARY 1993.

### 3. Modifiable data.

The following subparagraphs address the data contained in the data files which are read during initialization. Changes to the data files do not normally require a recompilation of the source code and relink of the libraries. The configuration management group should be contacted if it is necessary to make source code changes to the baseline.

#### 3.1 Aero\_data

This data array consists of characteristics and parameters describing the physical vehicle and its aerodynamic performance and control.

### 3.1.1 MOMENT\_OF\_INERTIA\_X

MOMENT\_OF\_INERTIA\_X is a constant defining the moment of inertia of the vehicle in the x-axis.

#### 3.1.1.1 Initialization

The constant MOMENT\_OF\_INERTIA\_X is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define MOMENT\_OF\_INERTIA\_X

aero\_data[0]

### 3.1.1.2 Usage

During real-time execution, this variable is not recomputed.

## 3.1.1.2.1 Algorithm

The MOMENT\_OF\_INERTIA\_X is used to initialize the [1][1] element of the inertia matrix in the CSU aerodyn\_init.

inertia\_matrix[1] [1] = MOMENT\_OF\_INERTIA\_X;

The MOMENT\_OF\_INERTIA\_X is used to compute forces in the CSC aerodyn\_simple\_simul.

See APPENDIX B for a complete source code listing.

## 3.1.2 MOMENT\_OF\_INERTIA\_Y

MOMENT\_OF\_INERTIA\_Y is a constant defining the moment of inertia of the vehicle in the y-axis.

#### 3.1.2.1 Initialization

The constant MOMENT\_OF\_INERTIA\_Y is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define MOMENT_OF_INERTIA_Y aero_data[ 1]
```

### 3.1.2.2 Usage

During real-time execution, this variable is not recomputed.

## 3.1.2.2.1 Algorithm

The MOMENT\_OF\_INERTIA\_Y is used to initialize the [2][2] element of the inertia matrix in the CSU aerodyn\_init.

```
inertia_matrix[2] [2] = MOMENT_OF_INERTIA_Y;
```

See APPENDIX B for a complete source code listing.

### 3.1.3 MOMENT\_OF\_INERTIA\_Z

MOMENT\_OF\_INERTIA\_Z is a constant defining the moment of inertia of the vehicle in the z-axis.

#### 3.1.3.1 Initialization

The constant MOMENT\_OF\_INERTIA\_Z is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define MOMENT\_OF\_INERTIA\_Z

aero\_data[2]

### 3.1.3.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.3.2.1 Algorithm

The MOMENT\_OF\_INERTIA\_Z is used to initialize the [3][3] element of the inertia matrix in the CSU aerodyn\_init.

inertia\_matrix[3] [3] = MOMENT\_OF\_INERTIA\_Z;

See APPENDIX B for a complete source code listing.

## 3.1.4 AIRFRAME\_MASS

AIRFRAME\_MASS is a constant defining the empty weight of the vehicle, especially, not including expendable items.

#### 3.1.4.1 Initialization

The constant AIRFRAME\_MASS is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define AIRFRAME\_MASS

aero\_data[3]

### 3.1.4.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.4.2.1 Algorithm

The AIRFRAME\_MASS is used to initialize the vehicle mass in the CSU aerodyn\_init.

The AIRFRAME\_MASS is used to update the vehicle gross weight by a call to the CSU compute\_gross\_weight.

```
vehicle_mass = AIRFRAME_MASS + ORDINANCE_MASS +
fuel_get_current_level() * KILOGRAMS_PER_GALLON;/* kg */
```

gross\_weight = vehicle\_mass \* GRAV\_CONSTANT; /\* N \*/

See APPENDIX B for a complete source code listing.

## 3.1.5 ORDÏNANCE\_MASS

ORDINANCE\_MASS is a constant defining the weight of the ordinance on board the vehicle, especially, the expendables.

#### 3.1.5.1 Initialization

The constant ORDINANCE\_MASS is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define ORDINANCE\_MASS

aero\_data[4]

### 3.1.5.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.5.2.1 Algorithm

The ORDINANCE\_MASS is used to initialize the vehicle mass in the CSU aerodyn\_init.

The ORDINANCE\_MASS is used to update the vehicle gross weight by a call to the CSU compute\_gross\_weight.

vehicle\_mass = AIRFRAME\_MASS + ORDINANCE\_MASS +
 fuel\_get\_current\_level() \* KILOGRAMS\_PER\_GALLON;/\* kg \*/

gross\_weight = vehicle\_mass \* GRAV\_CONSTANT; /\* N \*/

See APPENDIX B for a complete source code listing.

### 3.1.6 GRAV\_CONSTANT

GRAV\_CONSTANT is a constant defining the gravitational constant.

#### 3.1.6.1 Initialization

The constant GRAV\_CONSTANT is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define GRAV\_CONSTANT

aero\_data[5]

#### 3.1.6.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.6.2.1 Algorithm

The GRAV\_CONSTANT is used to update the vehicle gross weight by a call to the CSU compute\_gross\_weight.

See APPENDIX B for a complete source code listing.

### 3.1.7 CG\_AC\_X

CG\_AC\_X is a constant defining the location of the aircraft center of gravity in the x-axis.

#### 3.1.7.1 Initialization

The constant CG\_AC\_X is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

### 3.1.7.2 Usage

During real-time execution, this variable is not recomputed.

## 3.1.7.2.1 Algorithm

The CG\_AC\_X is used to initialize the vehicle location of the aircraft center of gravity in the x-axis in the CSU aerodyn\_init.

$$loc_ac_cg[X] = CG_AC_X;$$

See APPENDIX B for a complete source code listing.

#### 3.1.8 CG\_AC\_Y

CG\_AC\_Y is a constant defining the location of the aircraft center of gravity in the y-axis.

#### 3.1.8.1 Initialization

The constant CG\_AC\_Y is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define CG\_AC\_Y

aero data[7]

#### 3.1.8.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.8.2.1 Algorithm

The CG\_AC\_Y is used to initialize the vehicle location of the aircraft center of gravity in the y-axis in the CSU aerodyn\_init.

 $loc_ac_cg[Y] = CG_AC_Y;$ 

See APPENDIX B for a complete source code listing.

### 3.1.9 CG\_AC\_Z

CG\_AC\_Z is a constant defining the location of the aircraft center of gravity in the z-axis.

#### 3.1.9.1 Initialization

The constant CG\_AC\_Z is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

aero_data[ 8]
•

### 3.1.9.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.9.2.1 Algorithm

The CG\_AC\_Z is used to initialize the vehicle location of the aircraft center of gravity in the z-axis in the CSU aerodyn\_init.

$$loc_ac_cg[Z] = CG_AC_Z;$$

See APPENDIX B for a complete source code listing.

### 3.1.10 VIRTUAL\_WING\_AREA

VIRTUAL\_WING\_AREA is a constant defining the effective wing area of the vehicle.

#### 3.1.10.1 Initialization

The constant VIRTUAL\_WING\_AREA is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define	VIRTUAL_WING_AREA	aero_data[ 9]	

#### 3.1.10.2 Usage

During real-time execution, this variable is not recomputed.

## 3.1.10.2.1 Algorithm

The VIRTUAL\_WING\_AREA is used to compute the total lift of the virtual wing of the vehicle by a call to the CSU compute\_lift\_drag\_forces.

See APPENDIX B for a complete source code listing.

### 3.1.11 VIRTUAL\_WING\_COP\_AC\_X

VIRTUAL\_WING\_COP\_AC\_X is a constant defining the location in the x-axis of the virtual wing center of pressure.

#### 3.1.11.1 Initialization

The constant VIRTUAL\_WING\_COP\_AC\_X is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define VIRTUAL\_WING\_COP\_AC\_X

aero data[10]

### 3.1.11.2 Usage

During real-time execution, this variable is not recomputed.

## 3.1.11.2.1 Algorithm

The VIRTUAL\_WING\_COP\_AC\_X is initialize the location in the x-axis of the center of pressure for the virtual wing in the CSU aerodyn\_init.

loc\_ac\_virtual\_wing\_cop[X] = VIRTUAL\_WING\_COP\_AC\_X;

The location of the center of pressure for the virtual wing is used to compute lift and moments due to lift.

See APPENDIX B for a complete source code listing.

### 3.1.12 VIRTUAL\_WING\_COP\_AC\_Y

VIRTUAL\_WING\_COP\_AC\_Y is a constant defining the location in the y-axis of the virtual wing center of pressure.

#### 3.1.12.1 Initialization

The constant VIRTUAL\_WING\_COP\_AC\_Y is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define VIRTUAL\_WING\_COP\_AC\_Y

aero\_data[11]

### 3.1.12.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.12.2.1 Algorithm

The VIRTUAL\_WING\_COP\_AC\_Y is initialize the location in the y-axis of the center of pressure for the virtual wing in the CSU aerodyn\_init.

loc\_ac\_virtual\_wing\_cop[Y] = VIRTUAL\_WING\_COP\_AC\_Y;

The location of the center of pressure for the virtual wing is used to compute lift and moments due to lift.

See APPENDIX B for a complete source code listing.

## 3.1.13 VIRTUAL\_WING\_COP\_AC\_Z

VIRTUAL\_WING\_COP\_AC\_Z is a constant defining the location in the z-axis of the virtual wing center of pressure.

#### 3.1.13.1 Initialization

The constant VIRTUAL\_WING\_COP\_AC\_Z is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define VIRTUAL\_WING\_COP\_AC\_Z

aero\_data[12]

#### 3.1.13.2 Usage

During real-time execution, this variable is not recomputed.

#### 3.1.13.2.1 Algorithm

The VIRTUAL\_WING\_COP\_AC\_Z is initialize the location in the z-axis of the center of pressure for the virtual wing in the CSU aerodyn\_init.

loc\_ac\_virtual\_wing\_cop[Z] = VIRTUAL\_WING\_COP\_AC\_Z;

The location of the center of pressure for the virtual wing is used to compute lift and moments due to lift.

See APPENDIX B for a complete source code listing.

### 3.1.14 WING\_LIFT\_COEFFICIENT\_FIT\_3

WING\_LIFT\_COEFFICIENT\_FIT\_3 is a constant defining the fourth coefficient of the wing lift coefficient polynomial used to compute the wing lift coefficient.

#### 3.1.14.1 Initialization

The constant WING\_LIFT\_COEFFICIENT\_FIT\_3 is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn init is normally done only once during CSCI initialization and

is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define WING_LIFT_COEFFICIENT_FIT_3 aero_data[13]
```

#### 3.1.14.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.14.2.1 Algorithm

The WING\_LIFT\_COEFFICIENT\_FIT\_3 is used to compute the unit lift of the virtual wing by a call to the CSU virtual\_wing\_lift\_coefficient. The call to this CSU is commented out.

See APPENDIX B for a complete source code listing.

## 3.1.15 WING\_LIFT\_COEFFICIENT\_FIT\_2

WING\_LIFT\_COEFFICIENT\_FIT\_2 is a constant defining the third coefficient of the wing lift coefficient polynomial used to compute the wing lift coefficient.

#### 3.1.15.1 Initialization

The constant WING\_LIFT\_COEFFICIENT\_FIT\_2 is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define WING\_LIFT\_COEFFICIENT\_FIT\_2

aero\_data[14]

### 3.1.15.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.15.2.1 Algorithm

The WING\_LIFT\_COEFFICIENT\_FIT\_2 is used to compute the unit lift of the virtual wing by a call to the CSU virtual\_wing\_lift\_coefficient. The call to this CSU is commented out.

See APPENDIX B for a complete source code listing.

### 3.1.16 WING\_LIFT\_COEFFICIENT\_FIT\_1

WING\_LIFT\_COEFFICIENT\_FIT\_1 is a constant defining the second coefficient of the wing lift coefficient polynomial used to compute the wing lift coefficient.

#### 3.1.16.1 Initialization

The constant WING\_LIFT\_COEFFICIENT\_FIT\_1 is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define WING\_LIFT\_COEFFICIENT\_FIT\_1

aero\_data[15]

### 3.1.16.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.16.2.1 Algorithm

The WING\_LIFT\_COEFFICIENT\_FIT\_1 is used to compute the unit lift of the virtual wing by a call to the CSU virtual\_wing\_lift\_coefficient. The call to this CSU is commented out.

See APPENDIX B for a complete source code listing.

### 3.1.17 WING\_LIFT\_COEFFICIENT\_FIT\_0

WING\_LIFT\_COEFFICIENT\_FIT\_0 is a constant defining the first coefficient of the wing lift coefficient polynomial used to compute the wing lift coefficient.

### 3.1.17.1 Initialization

The constant WING\_LIFT\_COEFFICIENT\_FIT\_0 is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define WING_LIFT_COEFFICIENT_FIT_0 aero_data[16]
```

### 3.1.17.2 Usage

During real-time execution, this variable is not recomputed.

## 3.1.17.2.1 Algorithm

The WING\_LIFT\_COEFFICIENT\_FIT\_0 is used to compute the unit lift of the virtual wing by a call to the CSU virtual\_wing\_lift\_coefficient. The call to this CSU is commented out.

See APPENDIX B for a complete source code listing.

### 3.1.18 WING\_STALL\_AOA

WING\_STALL\_AOA is a constant defining the wing stall angle of attack.

#### 3.1.18.1 Initialization

The constant WING\_STALL\_AOA is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define WING_STALL_AOA (deg_to_rad(aero_data[17]))
```

### 3.1.18.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.18.2.1 Algorithm

The WING\_STALL\_AOA is used to control the computation of the unit lift coefficient of the virtual wing by a call to the CSU virtual\_wing\_lift\_coefficient. The call to this CSU is commented out.

See APPENDIX B for a complete source code listing.

### 3.1.19 VSTAB\_AREA

VSTAB\_AREA is a constant defining the effective vertical stabilator area.

#### 3.1.19.1 Initialization

The constant VSTAB\_AREA is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define VSTAB\_AREA

aero data[18]

### 3.1.19.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.19.2.1 Algorithm

VSTAB\_AREA is used to compute the total lift of the vertical stabilator by a call to CSU compute\_lift\_drag\_forces.

lift\_vstab = dynamic\_pressure \* lift\_coefficient\_vstab \* VSTAB\_AREA;

See APPENDIX B for a complete source code listing.

## 3.1.20 VSTAB\_COP\_AC\_X

VSTAB\_COP\_AC\_X is a constant defining the location in the x-axis of the center of pressure of the vertical stabilator for the vehicle.

### 3.1.20.1 Initialization

The constant VSTAB\_COP\_AC\_X is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define	VSTAB_	_COP_	_AC_	X

aero\_data[19]

### 3.1.20.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.20.2.1 Algorithm

VSTAB\_COP\_AC\_X is used to initialize the location in the x-axis of the center of pressure for the vertical stabilator in the CSU aerodyn\_init.

loc\_ac\_vstab\_cop[X] = VSTAB\_COP\_AC\_X;

The loc\_ac\_vstab\_cop vector is then used to compute the body forces and moments by a call to the CSC sum\_body\_forces\_and\_moments\_about\_ac.

vec\_cross\_prod(loc\_ac\_vstab\_cop, lift\_body\_vstab, moment\_body\_vstab);

See APPENDIX B for a complete source code listing.

## 3.1.21 VSTAB\_COP\_AC\_Y

VSTAB\_COP\_AC\_Y is a constant defining the location in the y-axis of the center of pressure of the vertical stabilator for the vehicle.

### 3.1.21.1 Initialization

The constant VSTAB\_COP\_AC\_Y is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define VSTAB\_COP\_AC\_Y

aero\_data[20]

## 3.1.21.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.21.2.1 Algorithm

VSTAB\_COP\_AC\_Y is used to initialize the location in the y-axis of the center of pressure for the vertical stabilator in the CSU aerodyn\_init.

loc\_ac\_vstab\_cop[Y] = VSTAB\_COP\_AC\_Y;

The loc\_ac\_vstab\_cop vector is then used to compute the body forces and moments by a call to the CSC sum\_body\_forces\_and\_moments\_about\_ac.

vec\_cross\_prod(loc\_ac\_vstab\_cop, lift\_body\_vstab, moment\_body\_vstab);

See APPENDIX B for a complete source code listing.

### 3.1.22 VSTAB\_COP\_AC\_Z

VSTAB\_COP\_AC\_Z is a constant defining the location in the z-axis of the center of pressure of the vertical stabilator for the vehicle.

#### 3.1.22.1 Initialization

The constant VSTAB\_COP\_AC\_Z is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define VSTAB\_COP\_AC\_Z

aero\_data[21]

## 3.1.22.2 Usage

During real-time execution, this variable is not recomputed.

## 3.1.22.2.1 Algorithm

VSTAB\_COP\_AC\_Z is used to initialize the location in the z-axis of the center of pressure for the vertical stabilator in the CSU aerodyn\_init.

loc\_ac\_vstab\_cop[Z] = VSTAB\_COP\_AC\_Z;

The loc\_ac\_vstab\_cop vector is then used to compute the body forces and moments by a call to the CSC sum\_body\_forces\_and\_moments\_about\_ac.

vec\_cross\_prod(loc\_ac\_vstab\_cop, lift\_body\_vstab, moment\_body\_vstab);

See APPENDIX B for a complete source code listing.

# 3.1.23 VSTAB\_LIFT\_COEFFICIENT\_1

VSTAB\_LIFT\_COEFFICIENT\_1 is a constant defining the second coefficient of the vertical stabilator coefficient polynomial.

## 3.1.23.1 Initialization

The constant VSTAB\_LIFT\_COEFFICIENT\_1 is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define VSTAB\_LIFT\_COEFFICIENT\_1

aero\_data[22]

## 3.1.23.2 Usage

During real-time execution, this variable is not recomputed.

## 3.1.23.2.1 Algorithm

VSTAB\_LIFT\_COEFFICIENT\_1 is used to compute the unit lift coefficient of the vertical stabilator in the CSU vstab\_lift\_coefficient.

```
if (abs(yaw) > VSTAB_STALL_SSA)
    yawval = sign(yawval) * VSTAB_STALL_SSA;
else
    yawval = yaw;
return (VSTAB_LIFT_COEFFICIENT_1 * yawval);
```

The vstab\_lift\_coefficient is used to compute the total vertical stabilator lift by a call to the CSU compute\_lift\_drag\_coefficients.

```
lift_coefficient_vstab = vstab_lift_coefficient (side_slip_angle);
```

See APPENDIX B for a complete source code listing.

### 3.1.24 VSTAB\_STALL\_SSA

VSTAB\_STALL\_SSA is a constant defining the stall angle of the vertical stabilator.

### 3.1.24.1 Initialization

The constant VSTAB\_STALL\_SSA is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define VSTAB\_STALL\_SSA (deg\_to\_rad(aero\_data[23]))

### 3.1.24.2 Usage

During real-time execution, this variable is not recomputed.

## 3.1.24.2.1 Algorithm

VSTAB\_STALL\_SSA is used to compute the unit lift coefficient of the vertical stabilator in the CSU vstab\_lift\_coefficient.

```
if (abs(yaw) > VSTAB_STALL_SSA)
    yawval = sign(yawval) * VSTAB_STALL_SSA;
else
    yawval = yaw;
return (VSTAB_LIFT_COEFFICIENT_1 * yawval);
```

The vstab\_lift\_coefficient is used to compute the total vertical stabilator lift by a call to the CSU compute\_lift\_drag\_coefficients.

```
lift_coefficient_vstab = vstab_lift_coefficient (side_slip_angle);
```

See APPENDIX B for a complete source code listing.

## 3.1.25 MAIN\_ROTOR\_COP\_AC\_X

MAIN\_ROTOR\_COP\_AC\_X is a constant defining the location in the x-axis of the center of pressure for the main rotor.

### 3.1.25.1 Initialization

The constant MAIN\_ROTOR\_COP\_AC\_X is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define MAIN\_ROTOR\_COP\_AC\_X aero\_data[24]

## 3.1.25.2 Usage

During real-time execution, this variable is not recomputed.

## 3.1.25.2.1 Algorithm

MAIN\_ROTOR\_COP\_AC\_X is used to initialize the location in the x-axis of the center of pressure for the main rotor in the CSU aerodyn\_init.

loc ac main\_rotor\_cop[X] = MAIN\_ROTOR\_COP\_AC\_X;

See APPENDIX B for a complete source code listing.

### 3.1.26 MAIN\_ROTOR\_COP\_AC\_Y

MAIN\_ROTOR\_COP\_AC\_Y is a constant defining the location in the y-axis of the center of pressure for the main rotor.

#### 3.1.26.1 Initialization

The constant MAIN\_ROTOR\_COP\_AC\_Y is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define MAIN\_ROTOR\_COP\_AC\_Y

aero\_data[25]

#### 3.1.26.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.26.2.1 Algorithm

MAIN\_ROTOR\_COP\_AC\_Y is used to initialize the location in the y-axis of the center of pressure for the main rotor in the CSU aerodyn\_init.

loc\_ac\_main\_rotor\_cop[Y] = MAIN\_ROTOR\_COP\_AC\_Y;

See APPENDIX B for a complete source code listing.

# 3.1.27 MAIN\_ROTOR\_COP\_AC\_Z

MAIN\_ROTOR\_COP\_AC\_Z is a constant defining the location in the z-axis of the center of pressure for the main rotor.

#### 3.1.27.1 Initialization

The constant MAIN\_ROTOR\_COP\_AC\_Z is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU

aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define MAIN\_ROTOR\_COP\_AC\_Z

aero\_data[26]

### 3.1.27.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.27.2.1 Algorithm

MAIN\_ROTOR\_COP\_AC\_Z is used to initialize the location in the z-axis of the center of pressure for the main rotor in the CSU aerodyn\_init.

loc\_ac\_main\_rotor\_cop[Z] = MAIN\_ROTOR\_COP\_AC\_Z;

See APPENDIX B for a complete source code listing.

# 3.1.28 MAIN\_ROTOR\_MAX\_THRUST

MAIN\_ROTOR\_MAX\_THRUST is a constant defining the maximum thrust of the main rotor at 100 per cent rpm.

#### 3.1.28.1 Initialization

The constant MAIN\_ROTOR\_MAX\_THRUST is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define MAIN\_ROTOR\_MAX\_THRUST

aero\_data[27]

# 3.1.28.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.28.2.1 Algorithm

MAIN\_ROTOR\_MAX\_THRUST is used to compute main\_rotor\_thrust in the CSU compute\_rotor\_forces\_and\_moments.

See APPENDIX B for a complete source code listing.

# 3.1.29 MAIN\_ROTOR\_MAST\_TILT

MAIN\_ROTOR\_MAST\_TILT is a constant defining the angle of tilt of the main rotor mast.

### 3,1.29.1 Initialization

The constant MAIN\_ROTOR\_MAST\_TILT is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define MAIN\_ROTOR\_MAST\_TILT (deg\_to\_rad(aero\_data[28]))

# 3.1.29.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.29.2.1 Algorithm

MAIN\_ROTOR\_MAST\_TILT is used to compute sine and cosine of the angle of main rotor mast tilt in the CSU aerodyn\_init.

```
MAIN_ROTOR_MAST_TILT_SIN = sin(MAIN_ROTOR_MAST_TILT);
MAIN_ROTOR_MAST_TILT_COS = cos(MAIN_ROTOR_MAST_TILT);
```

These values are used to compute the forces generated by the main rotor on the body by a call to the CSU compute\_rotor\_forces\_and\_moments.

See APPENDIX B for a complete source code listing.

# 3.1.30 MAIN\_ROTOR\_MAX\_LOAD\_TORQUE

MAIN\_ROTOR\_MAX\_LOAD\_TORQUE is a constant defining the maximum load torque of the main rotor.

#### 3.1.30.1 Initialization

The constant MAIN\_ROTOR\_MAX\_LOAD\_TORQUE is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define MAIN\_ROTOR\_MAX\_LOAD\_TORQUE

aero\_data[29]

### 3.1.30.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.30.2.1 Algorithm

MAIN\_ROTOR\_MAX\_LOAD\_TORQUE is used to compute the load torque of the main rotor by a call to the CSU compute\_rotor\_loads.

The main\_rotor\_load\_torque is used to compute the engine torque by a call to the CSU compute\_engine\_torque.

See APPENDIX B for a complete source code listing.

# 3.1.31 MAIN\_ROTOR\_MAX\_PITCH\_MOMENT

MAIN\_ROTOR\_MAX\_PITCH\_MOMENT is a constant defining the maximum pitching moment of the main rotor.

# 3.1.31.1 Initialization

The constant MAIN\_ROTOR\_MAX\_PITCH\_MOMENT is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define MAIN\_ROTOR\_MAX\_PITCH\_MOMENT

aero\_data[30]

# 3.1.31.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.31.2.1 Algorithm

MAIN\_ROTOR\_MAX\_PITCH\_MOMENT is used to compute the pitching moment on the body generated by the main rotor by a call to the CSU compute\_rotor\_forces\_and\_moments.

moment\_body\_main\_rotor[X] =
 - controller\_cyclic\_pitch \* MAIN\_ROTOR\_MAX\_PITCH\_MOMENT;

The components are summed for the total moment on the body by a call to the CSU sum\_body\_forces\_and\_moments\_about\_ac.

```
vec_init (moment_body);
vec_add (moment_body, moment_body_main_rotor, moment_body);
vec_add (moment_body, moment_body_tail_rotor, moment_body);
vec_add (moment_body, moment_body_virtual_wing, moment_body);
vec_add (moment_body, moment_body_vstab, moment_body);
vec_add (moment_body, moment_body_cg, moment_body);
vec_add (moment_body, ground_torque, moment_body);
vec_add (moment_body, moment_body_damping, moment_body);
```

# 3.1.32 MAIN\_ROTOR\_MAX\_ROLL\_MOMENT

MAIN\_ROTOR\_MAX\_ROLL\_MOMENT is a constant defining the maximum rolling moment of the main rotor.

#### 3.1.32.1 Initialization

The constant MAIN\_ROTOR\_MAX\_ROLL\_MOMENT is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define MAIN\_ROTOR\_MAX\_ROLL\_MOMENT aero\_data[31]

### 3.1.32.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.32.2.1 Algorithm

MAIN\_ROTOR\_MAX\_ROLL\_MOMENT is used to compute the rolling moment on the body generated by the main rotor by a call to the CSU compute\_rotor\_forces\_and\_moments.

```
moment_body_main_rotor[Y] =
  controller_cyclic_roll * MAIN_ROTOR_MAX_ROLL_MOMENT;
```

The components are summed for the total moment on the body by a call to the CSU sum\_body\_forces\_and\_moments\_about\_ac.

```
vec_init (moment_body);
vec_add (moment_body, moment_body_main_rotor, moment_body);
vec_add (moment_body, moment_body_tail_rotor, moment_body);
vec_add (moment_body, moment_body_virtual_wing, moment_body);
vec_add (moment_body, moment_body_vstab, moment_body);
vec_add (moment_body, moment_body_cg, moment_body);
vec_add (moment_body, ground_torque, moment_body);
vec_add (moment_body, moment_body_damping, moment_body);
```

# 3.1.33 MAIN\_ROTOR\_TORQUE\_COUPLING\_GAIN

MAIN\_ROTOR\_TORQUE\_COUPLING\_GAIN is a constant defining the torque moment generated by the main rotor.

#### 3.1.33.1 Initialization

The constant MAIN\_ROTOR\_TORQUE\_COUPLING\_GAIN is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define MAIN_ROTOR_TORQUE_COUPLING_GAIN aero_data[32]
```

#### 3.1.33.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.33.2.1 Algorithm

MAIN\_ROTOR\_TORQUE\_COUPLING\_GAIN is used to compute the torque moment on the body generated by the main rotor by a call to the CSU compute\_rotor\_forces\_and\_moments.

The components are summed for the total moment on the body by a call to the CSU sum\_body\_forces\_and\_moments\_about\_ac.

```
vec_init (moment_body);
vec_add (moment_body, moment_body_main_rotor, moment_body);
vec_add (moment_body, moment_body_tail_rotor, moment_body);
vec_add (moment_body, moment_body_virtual_wing, moment_body);
vec_add (moment_body, moment_body_vstab, moment_body);
vec_add (moment_body, moment_body_cg, moment_body);
vec_add (moment_body, ground_torque, moment_body);
vec_add (moment_body, moment_body_damping, moment_body);
```

See APPENDIX B for a complete source code listing.

# 3.1.34 MAIN\_ROTOR\_GROUND\_EFFECT\_FACTOR

MAIN\_ROTOR\_GROUND\_EFFECT\_FACTOR is a constant defining the ground effect on the main rotor.

#### 3.1.34.1 Initialization

The constant MAIN\_ROTOR\_GROUND\_EFFECT\_FACTOR is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define MAIN ROTOR_GROUND_EFFECT_FACTOR aero_data[33]
```

# 3.1.34.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.34.2.1 Algorithm

MAIN\_ROTOR\_GROUND\_EFFECT\_FACTOR is used to compute the ground effect force generated during proximity of the rotor and the ground by a call to the CSC interact\_with\_ground.

force\_ground\_effect[Z] = main\_rotor\_thrust
 \* MAIN\_ROTOR\_GROUND\_EFFECT\_FACTOR
 / (cig\_altitude\_above\_gnd() + 1.0);

See APPENDIX B for a complete source code listing.

# 3.1.35 TAIL\_ROTOR\_COP\_AC\_X

TAIL\_ROTOR\_COP\_AC\_X is a constant defining the location in the x-axis of the center of pressure of the tail rotor for the vehicle.

### 3.1.35.1 Initialization

The constant TAIL\_ROTOR\_COP\_AC\_X is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define TAIL\_ROTOR\_COP\_AC\_X

aero\_data[34]

# 3.1.35.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.35.2.1 Algorithm

TAIL\_ROTOR\_COP\_AC\_X is used to initialize the location in the x-axis of the center of pressure for the tail rotor by a call to the CSU aerodyn\_init.

loc\_ac\_tail\_rotor\_cop[X] = TAIL\_ROTOR\_COP\_AC\_X;

The loc\_ac\_tail\_rotor\_cop vector is then used to compute and sum the body forces and moments by a call to the CSU sum\_body\_forces\_and\_moments\_ac.

# 3.1.36 TAIL\_ROTOR\_COP\_AC\_Y

TAIL\_ROTOR\_COP\_AC\_Y is a constant defining the location in the y-axis of the center of pressure of the tail rotor for the vehicle.

#### 3.1.36.1 Initialization

The constant TAIL\_ROTOR\_COP\_AC\_Y is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define TAIL\_ROTOR\_COP\_AC\_Y

aero\_data[35]

### 3.1.36.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.36.2.1 Algorithm

TAIL\_ROTOR\_COP\_AC\_Y is used to initialize the location in the y-axis of the center of pressure for the tail rotor by a call to the CSU aerodyn\_init.

```
loc_ac_tail_rotor_cop[Y] = TAIL_ROTOR_COP_AC_Y;
```

The loc\_ac\_tail\_rotor\_cop vector is then used to compute and sum the body forces and moments by a call to the CSU sum\_body\_forces\_and\_moments\_ac.

See APPENDIX B for a complete source code listing.

# 3.1.37 TAIL\_ROTOR\_COP\_AC\_Z

TAIL\_ROTOR\_COP\_AC\_Z is a constant defining the location in the z-axis of the center of pressure of the tail rotor for the vehicle.

#### 3.1.37.1 Initialization

The constant TAIL\_ROTOR\_COP\_AC\_Z is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define TAIL\_ROTOR\_COP\_AC\_Z

aero\_data[36]

#### 3.1.37.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.37.2.1 Algorithm

TAIL\_ROTOR\_COP\_AC\_Z is used to initialize the location in the z-axis of the center of pressure for the tail rotor by a call to the CSU aerodyn\_init.

```
loc_ac_tail_rotor_cop[Z] = TAIL_ROTOR_COP_AC_Z;
```

The loc\_ac\_tail\_rotor\_cop vector is then used to compute and sum the body forces and moments by a call to the CSU sum\_body\_forces\_and\_moments\_ac.

See APPENDIX B for a complete source code listing.

# 3.1.38 TAIL\_ROTOR\_MAX\_THRUST

TAIL\_ROTOR\_MAX\_THRUST is a constant defining the maximum thrust of the tail rotor.

#### 3.1.38.1 Initialization

The constant TAIL\_ROTOR\_MAX\_THRUST is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU

aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define TAIL\_ROTOR\_MAX\_THRUST

aero\_data[37]

# 3.1.38.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.38.2.1 Algorithm

TAIL\_ROTOR\_MAX\_THRUST is used to compute the tail rotor thrust and its force on the body of the vehicle by a call to the CSU compute\_rotor\_forces\_and\_moments.

' tail\_rotor\_thrust = powertrain\_percent\_shaft\_speed \*
controller\_tail\_rotor \* TAIL\_ROTOR\_MAX\_THRUST;

force\_body\_tail\_rotor[X] = tail\_rotor\_thrust;

See APPENDIX B for a complete source code listing.

# 3.1.39 TAIL\_ROTOR\_MAX\_LOAD\_TORQUE

TAIL\_ROTOR\_MAX\_LOAD\_TORQUE is a constant defining the maximum load torque of the tail rotor at 100 percent rpm.

### 3.1.39.1 Initialization

The constant TAIL\_ROTOR\_MAX\_LOAD\_TORQUE is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define TAIL\_ROTOR\_MAX\_LOAD\_TORQUE

aero\_data[38]

# 3.1.39.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.39.2.1 Algorithm

TAIL\_ROTOR\_MAX\_LOAD\_TORQUE is used to load torque for the tail rotor by a call to the CSU compute\_rotor\_loads.

The tail\_rotor\_load\_torque is used to compute the engine torque by a call to the CSU compute\_engine\_torque.

See APPENDIX B for a complete source code listing.

## 3.1.40 P\_DRAG\_COEFF\_CONST

P\_DRAG\_COEFF\_CONST is one of five constants defining the coefficients of the parasitic drag profile cubic function used to compute the parasitic drag profile for the vehicle.

#### 3.1.40.1 Initialization

The constant P\_DRAG\_COEFF\_CONST is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define P\_DRAG\_COEFF\_CONST

aero\_data[39]

# 3.1.40.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.40.2.1 Algorithm

P\_DRAG\_COEFF\_CONST is used to compute the parasitic drag profile coefficient from a cubic function by a call to the CSU find\_cubic\_function.

The p\_drag\_fit\_coeff vector is initialized to zero, then computed. If an error, especially the result is 0.0, an error statement is generated. The p\_drag\_fit\_coeff vector is computed by a call to the CSU aerodyn\_init.

The p\_drag\_fit\_coeff vector is then used to compute the total incompressible\_drag\_coefficient by a call to the CSU compute\_lift\_drag\_coefficients.

The total drag is computed by a call to the CSU compute\_lift\_drag\_forces.

total\_drag = total\_incompressible\_drag\_coefficient \* dynamic\_pressure \* TOTAL WETTED\_SURFACE\_AREA;

See APPENDIX B for a complete source code listing.

### 3.1.41 P\_DRAG\_TAS\_BREAK -

P\_DRAG\_TAS\_BREAK is one of five constants defining the coefficients of the parasitic drag profile cubic function used to compute the parasitic drag profile for the vehicle.

#### 3.1.41.1 Initialization

The constant P\_DRAG\_TAS\_BREAK is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define P\_DRAG\_TAS\_BREAK

aero\_data[40]

# 3.1.41.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.41.2.1 Algorithm

See paragraph 3.1.40.

P\_DRAG\_TAS\_BREAK is used to control computation of the y-axis element of the drag force by a call to the CSU transform\_lift\_drag\_forces\_to\_body\_coordinates.

# 3.1.42 P\_DRAG\_COEFF\_BREAK

P\_DRAG\_COEFF\_BREAK is one of five constants defining the coefficients of the parasitic drag profile cubic function used to compute the parasitic drag profile for the vehicle.

#### 3.1.42.1 Initialization

The constant P\_DRAG\_COEFF\_BREAK is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define P\_DRAG\_COEFF\_BREAK aero\_data[41]

# 3.1.42.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.42.2.1 Algorithm

See paragraph 3.1.40.

See APPENDIX B for a complete source code listing.

#### 3.1.43 P\_DRAG\_TAS\_MAX

P\_DRAG\_TAS\_MAX is one of five constants defining the coefficients of the parasitic drag profile cubic function used to compute the parasitic drag profile for the vehicle.

#### 3.1.43.1 Initialization

The constant P\_DRAG\_TAS\_MAX is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define P\_DRAG\_TAS\_MAX

aero\_data[42]

#### 3.1.43.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.43.2.1 Algorithm

See paragraph 3.1.40.

See APPENDIX B for a complete source code listing.

### 3.1.44 P\_DRAG\_COEFF\_MAX

P\_DRAG\_COEFF\_MAX is one of five constants defining the coefficients of the parasitic drag profile cubic function used to compute the parasitic drag profile for the vehicle.

#### 3.1.44.1 Initialization

The constant P\_DRAG\_COEFF\_MAX is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define P\_DRAG\_COEFF\_MAX

aero\_data[43]

#### 3.1.44.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.44.2.1 Algorithm

See paragraph 3.1.40.

See APPENDIX B for a complete source code listing.

# 3.1.45 TOTAL\_WETTED\_SURFACE\_AREA

TOTAL\_WETTED\_SURFACE\_AREA is a constant defining the total wetted surface area of the vehicle.

### 3.1.45.1 Initialization

The constant TOTAL\_WETTED\_SURFACE\_AREA is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define TOTAL\_WETTED\_SURFACE\_AREA

aero\_data[44]

# 3.1.45.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.45.2.1 Algorithm

TOTAL\_WETTED\_SURFACE\_AREA is used to compute the total drag by a call to the CSU compute\_lift\_drag\_forces.

total\_drag = total\_incompressible\_drag\_coefficient \* dynamic\_pressure \* TOTAL\_WETTED\_SURFACE\_AREA;

See APPENDIX B for a complete source code listing.

# 3.1.46 MAX\_ATT\_CTL\_ANGLE\_STOP

MAX\_ATT\_CTL\_ANGLE\_STOP is a constant defining the maximum attitude control angle stop.

#### 3.1.46.1 Initialization

The constant MAX\_ATT\_CTL\_ANGLE\_STOP is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define MAX\_ATT\_CTL\_ANGLE\_STOP

aero\_data[45]

#### 3.1.46.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.46.2.1 Algorithm

MAX\_ATT\_CTL\_ANGLE\_STOP is used to limit the maximum attitude control angle for the simple flight mode by a call to the CSU compute\_stab\_augmentation\_gains.

```
#if ATT_DAMPING_MODE_SIMPLE
    if (true_airspeed > HOVER_SLOW_LIMIT )
        MAX_ATT_CTL_ANGLE =
            log( true_airspeed ) * MAX_ATT_DAMPING_FACTOR ;
    else if (true_airspeed < -HOVER_SLOW_LIMIT )
        MAX_ATT_CTL_ANGLE =
            log( -true_airspeed ) * MAX_ATT_DAMPING_FACTOR ;
    else
        MAX_ATT_CTL_ANGLE = MAX_ATT_CTL_ANGLE_STOP ;

MAX_ATT_CTL_ANGLE = deg_to_rad( MAX_ATT_CTL_ANGLE );
#endif</pre>
```

See APPENDIX B for a complete source code listing.

# 3.1.47 MAX\_ATT\_DAMPING\_FACTOR

MAX ATT DAMPING FACTOR is a constant defining the

#### 3.1.47.1 Initialization

The constant MAX\_ATT\_DAMPING\_FACTOR is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define MAX\_ATT\_DAMPING\_FACTOR aero\_data[46]

#### 3.1.47.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.47.2.1 Algorithm

MAX\_ATT\_DAMPING\_FACTOR is used to compute the maximum attitude control angle for the simple flight mode by a call to the CSU compute\_stab\_augmentation\_gains.

```
#if ATT_DAMPING_MODE_SIMPLE

if (true_airspeed > HOVER_SLOW_LIMIT )

MAX_ATT_CTL_ANGLE =

log( true_airspeed ) * MAX_ATT_DAMPING_FACTOR ;

else if (true_airspeed < -HOVER_SLOW_LIMIT )

MAX_ATT_CTL_ANGLE =

log( -true_airspeed ) * MAX_ATT_DAMPING_FACTOR ;

else

MAX_ATT_CTL_ANGLE = MAX_ATT_CTL_ANGLE_STOP ;

MAX_ATT_CTL_ANGLE = deg_to_rad( MAX_ATT_CTL_ANGLE );

#endif
```

See APPENDIX B for a complete source code listing.

# 3.1.48 HOVER\_SLOW\_LIMIT

HOVER\_SLOW\_LIMIT is a constant defining the slow limit speed in hover.

#### 3.1.48.1 Initialization

The constant HOVER\_SLOW\_LIMIT is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define HOVER_SLOW_LIMIT aero_data[47]
```

#### 3.1.48.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.48.2.1 Algorithm

HOVER\_SLOW\_LIMIT is used to control computation of the maximum attitude control angle for the simple flight mode by a call to the CSU compute\_stab\_augmentation\_gains.

```
#if ATT_DAMPING_MODE_SIMPLE
         if (true_airspeed < HOVER_SLOW_LIMIT)</pre>
            if (true airspeed > -HOVER_SLOW_LIMIT)
               MAX_ATT_CTL_ANGLE =
                   MAX_ATT_CTL_ANGLE_SLOW;
            else if (true airspeed > -HOVER MED LIMIT)
               MAX_ATT_CTL_ANGLE =
                   MAX_ATT_CTL_ANGLE_MED;
           else
               MAX ATT CTL ANGLE =
                    MAX ATT CTL ANGLE_NORM;
          else if (true airspeed < HOVER_MED_LIMIT)
               MAX_ATT_CTL_ANGLE =
                    MAX_ATT_CTL_ANGLE_MED;
          else
               MAX ATT CTL ANGLE =
                    MAX ATT CTL ANGLE_NORM;
#endif
```

```
#if ATT_DAMPING_MODE_SIMPLE
    if (true_airspeed > HOVER_SLOW_LIMIT )
        MAX_ATT_CTL_ANGLE =
        log( true_airspeed ) * MAX_ATT_DAMPING_FACTOR ;
    else if (true_airspeed < -HOVER_SLOW_LIMIT )
        MAX_ATT_CTL_ANGLE =
        log( -true_airspeed ) * MAX_ATT_DAMPING_FACTOR ;
    else
        MAX_ATT_CTL_ANGLE =
            MAX_ATT_CTL_ANGLE =
            MAX_ATT_CTL_ANGLE =
            deg_to_rad( MAX_ATT_CTL_ANGLE );
#endif
```

# 3.1.49 HOVER\_AUG\_PITCH\_RESET\_VALUE

HOVER\_AUG\_PITCH\_RESET\_VALUE is a constant defining the value for the hover augmentation pitch integrator is reset when hover hold is turned on.

#### 3.1.49.1 Initialization

The constant HOVER\_AUG\_PITCH\_RESET\_VALUE is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define HOVER\_AUG\_PITCH\_RESET\_VALUE aero\_data[48]

# 3.1.49.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.49.2.1 Algorithm

HOVER\_AUG\_PITCH\_RESET\_VALUE is used to compute the hover augmentation pitch integrator by a call to the CSU compute\_stab\_augmentation\_gains.

```
if (hover_hold_state == ON)
     if (!hover_hold_turned_on )
           hover_hold_turned_on = TRUE;
           /* You should already be "hovering" (airspeed < 10 knots)
             for hover hold to show little visible swaying. */
           hover_aug_roll_integrator = 0.0;
           hover_aug_pitch_integrator =
                 HOVER_AUG_PITCH_RESET_VALUE;
     hover_aug_pitch_integrator +=
                 HOVER_AUG_PITCH_I_GAIN * velocity_vector[Y];
     hover_aug_pitch_integrator =
                 limiter(-0.2,hover_aug_pitch_integrator,0.2);
     hover_aug_pitch_angle = HOVER_AUG_PITCH_P_GAIN *
                 velocity_vector[Y]
                 + hover_aug_pitch_integrator;
     hover_aug_pitch_angle = limiter (-MAX_ATT_CTL_ANGLE,
                               hover_aug_pitch_angle,
                               MAX ATT CTL ANGLE);
 }
 else
#ifdef notdef
                                         /* added 8/31/89 (jwc) */
     hover aug roll integrator = 0.0;
     hover_aug_pitch_integrator = 0.0;
#endif
 }
```

# 3.1.50 MAX\_ATT\_CTL\_ANGLE\_NORM

MAX\_ATT\_CTL\_ANGLE\_NORM is a constant defining the normal setting for the maximum attitude control angle.

#### 3.1.50.1 Initialization

The constant MAX\_ATT\_CTL\_ANGLE\_NORM is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define MAX_ATT_CTL_ANGLE_NORM (deg_to_rad (aero_data[49]))
```

#### 3.1.50.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.50.2.1 Algorithm

MAX\_ATT\_CTL\_ANGLE\_NORM is used to initialize the maximum attitude control angle to the normal setting for the simple flight mode by a call to the CSU compute\_stab\_augmentation\_gains.

```
if ( !hover_hold_turned_on )
          hover_hold_turned_on = TRUE;
#if ATT_DAMPING_MODE_SIMPLE
          if (true_airspeed < HOVER_SLOW_LIMIT)</pre>
            if (true_airspeed > -HOVER_SLOW_LIMIT)
               MAX_ATT_CTL_ANGLE =
                         MAX ATT CTL_ANGLE_SLOW;
            else if (true_airspeed > -HOVER_MED_LIMIT)
               MAX ATT CTL ANGLE =
                         MAX ATT CTL_ANGLE_MED;
            else
               MAX_ATT_CTL_ANGLE =
                         MAX_ATT_CTL_ANGLE_NORM;
          else if (true_airspeed < HOVER_MED_LIMIT)
               MAX_ATT_CTL_ANGLE =
                         MAX_ATT_CTL_ANGLE_MED;
```

```
else

MAX_ATT_CTL_ANGLE =

MAX_ATT_CTL_ANGLE_NORM ;

#endif
}
```

# 3.1.51 MAX\_ATT\_CTL\_ANGLE\_MED

MAX\_ATT\_CTL\_ANGLE\_MED is a constant defining the medium setting for the maximum attitude control angle.

# 3.1.51.1 Initialization

The constant MAX\_ATT\_CTL\_ANGLE\_MED is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define MAX_ATT_CTL_ANGLE_MED (deg_to_rad (aero_data[50]))
```

# 3.1.51.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.51.2.1 Algorithm

MAX\_ATT\_CTL\_ANGLE\_MED is used to initialize the maximum attitude control angle to the normal setting for the simple flight mode by a call to the CSU compute\_stab\_augmentation\_gains.

```
MAX_ATT_CTL ANGLE =
                       MAX ATT CTL ANGLE_SLOW;
           else if (true airspeed > -HOVER_MED_LIMIT)
              MAX ATT_CTL_ANGLE =
                       MAX_ATT_CTL_ANGLE_MED;
           else
              MAX_ATT_CTL_ANGLE =
                       MAX_ATT_CTL_ANGLE_NORM;
         else if (true_airspeed < HOVER_MED_LIMIT)
              MAX_ATT_CTL_ANGLE =
                        MAX ATT CTL ANGLE MED;
         else
              MAX ATT CTL_ANGLE =
                        MAX_ATT_CTL_ANGLE_NORM;
#endif
         }
```

# 3.1.52 MAX\_ATT\_CTL\_ANGLE\_SLOW

MAX\_ATT\_CTL\_ANGLE\_SLOW is a constant defining the slow setting for the maximum attitude control angle.

#### 3.1.52.1 Initialization

The constant MAX\_ATT\_CTL\_ANGLE\_SLOW is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define MAX_ATT_CTL_ANGLE_SLOW (deg_to_rad (aero_data[51]))
```

# 3.1.52.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.52.2.1 Algorithm

MAX\_ATT\_CTL\_ANGLE\_SLOW is used to initialize the maximum attitude control angle to the slow setting for the simple flight mode by a call to the CSU compute\_stab\_augmentation\_gains.

```
if ( !hover_hold_turned_on )
          hover_hold_turned_on = TRUE;
#if ATT_DAMPING_MODE_SIMPLE
          if (true_airspeed < HOVER_SLOW_LIMIT)</pre>
            if (true airspeed > -HOVER_SLOW_LIMIT)
               MAX_ATT_CTL_ANGLE =
                         MAX_ATT_CTL_ANGLE_SLOW;
            else if (true_airspeed > -HOVER_MED_LIMIT)
               MAX_ATT_CTL_ANGLE =
                         MAX_ATT_CTL_ANGLE_MED;
            else
               MAX_ATT_CTL_ANGLE =
                         MAX_ATT_CTL_ANGLE_NORM;
          else if (true_airspeed < HOVER_MED_LIMIT)
               MAX ATT CTL ANGLE =
                         MAX_ATT_CTL_ANGLE_MED;
          else
               MAX ATT CTL ANGLE =
                         MAX_ATT_CTL_ANGLE_NORM;
#endif
          }
```

# 3.1.53 HOVER\_MED\_LIMIT

HOVER\_MED\_LIMIT is a constant defining the medium speed limit for hover.

#### 3.1.53.1 Initialization

The constant HOVER\_MED\_LIMIT is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define HOVER_MED_LIMIT aero_data[52]
```

## 3.1.53.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.53.2.1 Algorithm

HOVER\_MED\_LIMIT is used to control the computation of the maximum attitude control angle in the simple flight mode by a call to the CSU compute\_stab\_augmentation\_gains.

```
if (hover_hold_state == ON)
     if (!hover_hold_turned_on )
. .'. .
#if ATT_DAMPING_MODE_SIMPLE
          if (true_airspeed < HOVER_SLOW_LIMIT)</pre>
            if (true_airspeed > -HOVER_SLOW_LIMIT)
               MAX_ATT_CTL_ANGLE =
                    MAX_ATT_CTL_ANGLE_SLOW;
            else if (true_airspeed > -HOVER_MED_LIMIT)
               MAX_ATT_CTL_ANGLE =
                    MAX_ATT_CTL_ANGLE_MED;
            else
               MAX_ATT_CTL_ANGLE =
                    MAX_ATT_CTL_ANGLE_NORM;
          else if (true_airspeed < HOVER_MED_LIMIT)
               MAX_ATT_CTL_ANGLE =
                    MAX ATT CTL ANGLE_MED;
          else
               MAX ATT CTL ANGLE =
               MAX_ATT_CTL_ANGLE_NORM;
#endif
```

See APPENDIX B for a complete source code listing.

### 3.1.54 ATT\_CTL\_PITCH\_P\_GAIN

ATT\_CTL\_PITCH\_P\_GAIN is a constant defining the slope for the attitude control pitch command equation.

#### 3.1.54.1 Initialization

The constant ATT\_CTL\_PITCH\_P\_GAIN is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define ATT\_CTL\_PITCH\_P\_GAIN

aero\_data[53]

#### 3.1.54.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.54.2.1 Algorithm

ATT\_CTL\_PITCH\_P\_GAIN is used to compute the attitude control pitch command by a call to the CSU set\_pitch\_attitude.

See APPENDIX B for a complete source code listing.

# 3.1.55 ATT\_CTL\_PITCH\_I\_GAIN

ATT\_CTL\_PITCH\_I\_GAIN is a constant defining the slope for the attitude control pitch integrator equation.

#### 3.1.55.1 Initialization

The constant ATT\_CTL\_PITCH\_I\_GAIN is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define ATT\_CTL\_PITCH\_I\_GAIN

aero\_data[54]

### 3.1.55.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.55.2.1 Algorithm

ATT\_CTL\_PITCH\_I\_GAIN is used to compute the attitude control pitch integrator by a call to the CSU set\_pitch\_attitude. The integrator is used to compute the attitude control pitch command.

See APPENDIX B for a complete source code listing.

# 3.1.56 ATT\_CTL\_ROLL\_P\_GAIN

ATT\_CTL\_ROLL\_P\_GAIN is a constant defining the slope for the attitude control roll command equation.

#### 3.1.56.1 Initialization

The constant ATT\_CTL\_ROLL\_P\_GAIN is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define ATT_CTL_ROLL_P_GAIN aero_data[55]
```

#### 3.1.56.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.56.2.1 Algorithm

ATT\_CTL\_ROLL\_P\_GAIN is used to compute the attitude control roll command by a call to the CSU set\_roll\_attitude.

```
attitude_control_roll_integrator += ATT_CTL_ROLL_I_GAIN *
         (roll - angle);
/**** These used to be attitude control_pitch_integrator instead of
                                      PJM 11-1-89
    attitude_control_roll_integrator.
attitude control pitch_integrator =
         limiter (-0.1, attitude_control_pitch_integrator, 0.1);
***** /
attitude_control_roll_integrator =
         limiter (-0.1, attitude_control_roll_integrator, 0.1);
attitude control roll command = ATT_CTL_ROLL_P_GAIN *
         (roll - angle);
attitude control roll command += attitude_control_roll_integrator;
attitude_control_roll_command = limiter (
         -MAX_STAB_AUG_PITCH_ROLL_CONTROL,
         attitude_control_roll_command,
         MAX_STAB_AUG_PITCH_ROLL_CONTROL);
return (attitude_control_roll_command);
```

See APPENDIX B for a complete source code listing.

### 3.1.57 ATT\_CTL\_ROLL\_I\_GAIN

ATT\_CTL\_ROLL\_I\_GAIN is a constant defining the slope for the attitude control roll integrator equation.

#### 3.1.57.1 Initialization

The constant ATT\_CTL\_ROLL\_I\_GAIN is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define ATT\_CTL\_ROLL\_I\_GAIN

aero\_data[56]

### 3.1.57.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.57.2.1 Algorithm

ATT\_CTL\_ROLL\_I\_GAIN is used to compute the attitude control roll integrator by a call to the CSU set\_roll\_attitude. The integrator is used to compute the attitude control roll command.

```
attitude_control_roll_integrator += ATT_CTL_ROLL_I_GAIN *
        (roll - angle);
/**** These used to be attitude_control_pitch_integrator instead of
    attitude_control_roll_integrator.
                                   PIM 11-1-89
attitude_control_pitch_integrator =
        limiter (-0.1, attitude control_pitch_integrator, 0.1);
attitude_control_roll_integrator =
        limiter (-0.1, attitude control_roll_integrator, 0.1);
attitude control_roll_command = ATT_CTL_ROLL_P_GAIN *
         (roll - angle);
attitude control roll command += attitude_control_roll_integrator;
attitude_control_roll_command = limiter (
        -MAX_STAB_AUG_PITCH_ROLL_CONTROL,
         attitude control roll command,
         MAX_STAB_AUG_PITCH_ROLL_CONTROL);
return (attitude_control_roll_command);
```

## 3.1.58 HOVER\_AUG\_ROLL\_P\_GAIN

HOVER\_AUG\_ROLL\_P\_GAIN is a constant defining the slope for the hover augmentation roll angle equation.

#### 3.1.58.1 Initialization

The constant HOVER\_AUG\_ROLL\_P\_GAIN is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define HOVER_AUG_ROLL_P_GAIN aero_data[57]
```

### 3.1.58.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.58.2.1 Algorithm

HOVER\_AUG\_ROLL\_P\_GAIN is used to compute the hover augmentation roll angle by a call to the CSU compute\_stab\_augmentation\_gains.

# 3.1.59 HOVER\_AUG\_ROLL\_I\_GAIN

HOVER\_AUG\_ROLL\_I\_GAIN is a constant defining the slope for the hover augmentation roll integrator equation.

#### 3.1.59.1 Initialization

The constant HOVER\_AUG\_ROLL\_I\_GAIN is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define HOVER_AUG_ROLL_I_GAIN aero_data[58]
```

# 3.1.59.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.59.2.1 Algorithm

HOVER\_AUG\_ROLL\_I\_GAIN is used to compute the hover augmentation roll integrator by a call to the CSU compute\_stab\_augmentation\_gains. The integrator is used to compute the hover augmentation roll angle.

```
if (hover_hold_state == ON)
     if ( !hover_hold_turned_on )
           hover_hold_turned_on = TRUE;
           /* You should already be "hovering" (airspeed < 10 knots)
             for hover hold to show little visible swaying. */
           hover_aug_roll_integrator = 0.0;
     hover_aug_roll_integrator +=
                 HOVER AUG ROLL_I_GAIN * velocity_vector[X];
     hover_aug_roll_integrator =
                 limiter(-0.2,hover_aug_roll_integrator,0.2);
     hover_aug_roll_angle = HOVER_AUG_ROLL_P_GAIN *
                              velocity_vector[X]
                  + hover_aug_roll_integrator;
     hover_aug_roll_angle = limiter (-MAX_ATT_CTL_ANGLE,
                              hover aug roll angle,
                              MAX ATT CTL_ANGLE);
     stab_aug_roll = set_roll_attitude (hover_aug_roll_angle);
 }
  else
      stab_aug_roll = 0.0;
#ifdef notdef
      hover_aug_roll_integrator = 0.0;
                                          /* added 8/31/89 (jwc) */
      hover_aug_pitch_integrator = 0.0;
#endif
   controller_cyclic_roll = cyclic_roll + stab_aug_roll;
```

## 3.1.60 HOVER\_AUG\_PITCH\_P\_GAIN

HOVER\_AUG\_PITCH\_P\_GAIN is a constant defining the slope for the hover augmentation pitch angle equation.

### 3.1.60.1 Initialization

The constant HOVER\_AUG\_PITCH\_P\_GAIN is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define HOVER_AUG_PITCH_P_GAIN aero_data[59]
```

### 3.1.60.2 Usage

During real-time execution, this variable is not recomputed.

## 3.1.60.2.1 Algorithm

HOVER\_AUG\_PITCH\_P\_GAIN is used to compute the hover augmentation pitch angle by a call to the CSU compute\_stab\_augmentation\_gains.

## 3.1.61 HOVER\_AUG\_PITCH\_I\_GAIN

HOVER\_AUG\_PITCH\_I\_GAIN is a constant defining the slope for the hover augmentation pitch integrator equation.

#### 3.1.61.1 Initialization

The constant HOVER\_AUG\_PITCH\_I\_GAIN is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define HOVER\_AUG\_PITCH\_I\_GAIN aero\_data[60]

## 3.1.61.2 Usage

During real-time execution, this variable is not recomputed.

## 3.1.61.2.1 Algorithm

HOVER\_AUG\_PITCH\_I\_GAIN is used to compute the hover augmentation pitch integrator by a call to the CSU compute\_stab\_augmentation\_gains. The integrator is used to compute the hover augmentation pitch angle.

```
if (hover_hold_state == ON)
     if (!hover_hold_turned_on )
           hover_aug_pitch_integrator =
                       HOVER AUG PITCH RESET_VALUE;
     hover_aug_pitch_integrator +=
                 HOVER_AUG_PITCH_I_GAIN * velocity_vector[Y];
     hover_aug_pitch_integrator =
                 limiter(-0.2,hover_aug_pitch_integrator,0.2);
     hover_aug_pitch_angle = HOVER_AUG_PITCH_P_GAIN *
                 velocity_vector[Y]
                 + hover_aug_pitch_integrator;
     hover_aug_pitch_angle = limiter (-MAX_ATT_CTL_ANGLE,
                               hover aug pitch_angle,
                               MAX ATT_CTL_ANGLE);
     stab_aug_pitch = set_pitch_attitude (hover_aug_pitch_angle);
 else
     stab_aug_pitch = 0.0;
#ifdef notdef
                                        /* added 8/31/89 (jwc) */
     hover_aug_roll_integrator = 0.0;
     hover_aug_pitch_integrator = 0.0;
#endif
  controller_cyclic_pitch = cyclic_pitch + stab_aug_pitch;
```

## 3.1.62 HOVER\_AUG\_YAW\_P\_GAIN

HOVER\_AUG\_YAW\_P\_GAIN is a constant defining the slope for the hover augmentation yaw angle equation.

#### 3.1.62.1 Initialization

The constant HOVER\_AUG\_YAW\_P\_GAIN is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define HOVER_AUG_YAW_P_GAIN aero_data[61]
```

### 3.1.62.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.62.2.1 Algorithm

HOVER\_AUG\_YAW\_P\_GAIN is used to compute the hover augmentation yaw angle by a call to the CSU compute\_stab\_augmentation\_gains.

```
else
{
....
    stab_aug_yaw = 0.0;
....
}
....
controller_tail_rotor = pedal + stab_aug_yaw;
....
```

### 3.1.63 HOVER\_AUG\_YAW\_I\_GAIN

HOVER\_AUG\_YAW\_I\_GAIN is a constant defining the slope for the hover augmentation yaw integrator equation.

#### 3.1.63.1 Initialization

The constant HOVER\_AUG\_YAW\_I\_GAIN is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define HOVER_AUG_YAW_I_GAIN aero_data[62]
```

## 3.1.63.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.63.2.1 Algorithm

HOVER\_AUG\_YAW\_I\_GAIN is used to compute the hover augmentation yaw integrator by a call to the CSU compute\_stab\_augmentation\_gains. The integrator is used to compute the hover augmentation yaw angle.

```
if (hover_hold_state == ON)
{
    if ( !hover_hold_turned_on )
      {
    ....
```

#### 3.1.64.2 Usage

During real-time execution, this variable is not recomputed.

## 3.1.64.2.1 Algorithm

HOVER\_AUG\_CLIMB\_P\_GAIN is used to compute the hover augmentation climb angle by a call to the CSU compute\_stab\_augmentation\_gains.

```
if (hover_hold_state == ON)
   if (!hover_hold_turned_on )
        stab aug climb integrator = 0.0;
   stab_aug_climb_integrator -=
              HOVER_AUG_CLIMB_I_GAIN * velocity_vector[Z];
   if (stab aug climb integrator > 0.2) stab aug climb_integrator = 0.2;
   if (stab_aug_climb_integrator < -0.2) stab_aug_climb_integrator = -0.2;
   stab_aug_climb = - HOVER_AUG_CLIMB_P_GAIN *
                    velocity vector[Z] + stab_aug_climb_integrator;
   stab_aug_yaw = limiter (
                    -MAX_STAB_AUG_YAW_CLIMB_CONTROL,
                    stab_aug_yaw,
                     MAX_STAB_AUG_YAW_CLIMB_CONTROL);
   stab aug climb = limiter (
                    -MAX_STAB_AUG_YAW_CLIMB_CONTROL,
                    stab aug climb,
                     MAX STAB AUG YAW CLIMB CONTROL);
```

```
else
{
....
    stab_aug_climb = 0.0;
....
}
....
controller_collective = collective + stab_aug_climb;
```

## 3.1.65 HOVER\_AUG\_CLIMB\_I\_GAIN

HOVER\_AUG\_CLIMB\_I\_GAIN is a constant defining the slope for the hover augmentation climb integrator equation.

### 3.1.65.1 Initialization

The constant HOVER\_AUG\_CLIMB\_I\_GAIN is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define HOVER\_AUG\_CLIMB\_I\_GAIN aero\_data[64]

## 3.1.65.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.65.2.1 Algorithm

HOVER\_AUG\_CLIMB\_I\_GAIN is used to compute the hover augmentation climb integrator by a call to the CSU compute\_stab\_augmentation\_gains. The integrator is used to compute the hover augmentation climb angle.

```
if (hover hold_state == ON)
   if (!hover_hold_turned_on )
         stab_aug_climb_integrator = 0.0;
   stab_aug_climb_integrator -=
               HOVER_AUG_CLIMB_I_GAIN * velocity_vector[Z];
   if (stab aug_climb_integrator > 0.2) stab_aug_climb_integrator = 0.2;
   if (stab_aug_climb_integrator < -0.2) stab_aug_climb_integrator = -0.2;
   stab_aug_climb = - HOVER_AUG_CLIMB_P_GAIN *
                     velocity vector[Z] + stab_aug_climb_integrator;
   stab_aug_yaw = limiter (
                     -MAX_STAB_AUG_YAW_CLIMB_CONTROL,
                     stab_aug_yaw,
                      MAX_STAB_AUG_YAW_CLIMB_CONTROL);
    stab aug climb = limiter (
                     -MAX_STAB_AUG_YAW_CLIMB_CONTROL,
                      stab aug climb,
                      MAX_STAB_AUG_YAW_CLIMB_CONTROL);
}
else
    stab_aug_climb = 0.0;
}
controller_collective = collective + stab_aug_climb;
```

# 3.1.66 MAX\_STAB\_AUG\_PITCH\_ROLL\_CONTROL

MAX\_STAB\_AUG\_PITCH\_ROLL\_CONTROL is a constant defining the upper and lower limits of the attitude control roll command and attitude control pitch command for cross coupling effects.

#### 3.1.66.1 Initialization

The constant MAX\_STAB\_AUG\_PITCH\_ROLL\_CONTROL is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define MAX_STAB_AUG_PITCH_ROLL_CONTROL aero_data[65]
```

### 3.1.66.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.66.2.1 Algorithm

MAX\_STAB\_AUG\_PITCH\_ROLL\_CONTROL is used to compute the upper and lower limits of the attitude control roll command by a call to the CSU set\_roll\_attitude.

```
attitude_control_roll_integrator += ATT_CTL_ROLL_I_GAIN *
               (roll - angle);
/**** These used to be attitude control pitch integrator instead of
    attitude_control_roll_integrator.
                                      PIM 11-1-89
attitude_control_pitch_integrator =
               limiter (-0.1, attitude_control_pitch_integrator, 0.1);
***** /
attitude control roll integrator =
         limiter (-0.1, attitude_control_roll_integrator, 0.1);
attitude control roll command = ATT_CTL_ROLL_P_GAIN *
               (roll - angle);
attitude_control_roll_command += attitude_control_roll_integrator;
attitude_control_roll_command = limiter (
               -MAX_STAB_AUG_PITCH_ROLL_CONTROL,
               attitude control roll_command,
               MAX STAB AUG PITCH_ROLL_CONTROL);
return (attitude_control_roll_command);
```

MAX\_STAB\_AUG\_PITCH\_ROLL\_CONTROL is used to compute the upper and lower limits of the attitude control pitch command by a call to the CSU set\_pitch\_attitude.

# 3.1.67 MAX\_STAB\_AUG\_YAW\_CLIMB\_CONTROL

MAX\_STAB\_AUG\_YAW\_CLIMB\_CONTROL is a constant defining the upper and lower limits of the stabilator augmentation yaw command and stabilator augmentation climb command for cross coupling effects.

#### 3.1.67.1 Initialization

The constant MAX\_STAB\_AUG\_YAW\_CLIMB\_CONTROL is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define MAX\_STAB\_AUG\_YAW\_CLIMB\_CONTROL aero\_data[66]

## 3.1.67.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.67.2.1 Algorithm

MAX\_STAB\_AUG\_YAW\_CLIMB\_CONTROL is used to compute the upper and lower limits of the stabilator augmentation yaw command and stabilator augmentation climb command by a call to the CSU compute stab\_augmentation\_gains.

```
if (hover_hold_state == ON)
   if ( !hover_hold_turned_on )
         stab_aug_climb_integrator = 0.0;
   stab_aug_climb_integrator -=
               HOVER_AUG_CLIMB_I_GAIN * velocity_vector[Z];
   if (stab aug climb integrator > 0.2) stab_aug_climb_integrator = 0.2;
   if (stab aug climb_integrator < -0.2) stab_aug_climb_integrator = -0.2;
   stab aug climb = - HOVER_AUG_CLIMB_P_GAIN *
                     velocity_vector[Z] + stab_aug_climb_integrator;
   stab_aug_yaw = limiter (
                     -MAX_STAB_AUG_YAW_CLIMB_CONTROL,
                     stab_aug_yaw,
                      MAX_STAB_AUG_YAW_CLIMB_CONTROL);
   stab aug climb = limiter (
                     -MAX_STAB_AUG_YAW_CLIMB_CONTROL,
                     stab aug climb,
                      MAX_STAB_AUG_YAW_CLIMB_CONTROL);
}
else
   stab_aug_climb = 0.0;
}
controller_collective = collective + stab_aug_climb;
```

# 3.1.68 ROLL\_RATE\_DAMPING\_GAIN

ROLL\_RATE\_DAMPING\_GAIN is a constant defining the roll damping rate.

#### 3.1.68.1 Initialization

The constant ROLL\_RATE\_DAMPING\_GAIN is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define ROLL_RATE_DAMPING_GAIN aero_data[67]
```

### 3.1.68.2 Usage

During real-time execution, this variable is not recomputed.

## 3.1.68.2.1 Algorithm

ROLL\_RATE\_DAMPING\_GAIN is used to initialize the roll damping by a call to the CSU aerodyn\_init.

```
roll_damping = ROLL_RATE_DAMPING_GAIN;
```

The roll damping is increased if hover hold is turned on by a call to the CSU compute\_stab\_augmentation\_gains.

```
if (hover_hold_state == ON)
{
    if ( !hover_hold_turned_on )
        {
        hover_hold_turned_on = TRUE ;

        roll_damping = 2 * ROLL_RATE_DAMPING_GAIN;
....
    }
....
}
```

```
else
{
....
roll_damping = ROLL_RATE_DAMPING_GAIN;
....
}
```

The roll\_damping is used to compute the y-axis element of damping of the body moment vector by a call to the CSU compute\_body\_damping\_forces\_and\_moments.

```
moment_body_damping[Y] = - roll_damping * roll_rate;
```

See APPENDIX B for a complete source code listing.

# 3.1.69 PITCH\_RATE\_DAMPING\_GAIN

PITCH\_RATE\_DAMPING\_GAIN is a constant defining the pitch damping rate.

### 3.1.69.1 Initialization

The constant PITCH\_RATE\_DAMPING\_GAIN is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

```
#define PITCH_RATE_DAMPING_GAIN aero_data[68]
```

## 3.1.69.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.69.2.1 Algorithm

PITCH\_RATE\_DAMPING\_GAIN is used to initialize the pitch damping by a call to the CSU aerodyn\_init.

```
pitch_damping = PITCH_RATE_DAMPING_GAIN;
```

The pitch damping is increased if hover hold is turned on by a call to the CSU compute\_stab\_augmentation\_gains.

The pitch\_damping is used to compute the x-axis element of damping of the body moment vector by a call to the CSU compute\_body\_damping\_forces\_and\_moments.

```
moment_body_damping[X] = - pitch_damping * pitch_rate;
```

See APPENDIX B for a complete source code listing.

# 3.1.70 YAW\_RATE\_DAMPING\_GAIN

YAW\_RATE\_DAMPING\_GAIN is a constant defining the yaw damping rate.

### 3.1.70.1 Initialization

The constant YAW\_RATE\_DAMPING\_GAIN is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU

Reference # W003092; 31 March 1993 VOLUME 1 of 3; Rev. 0.0

aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define YAW\_RATE\_DAMPING\_GAIN

aero\_data[69]

### 3.1.70.2 Usage

During real-time execution, this variable is not recomputed.

## 3.1.70.2.1 Algorithm

YAW\_RATE\_DAMPING\_GAIN is used to initialize the yaw damping by a call to the CSU aerodyn\_init.

yaw\_damping = YAW\_RATE\_DAMPING\_GAIN;

The yaw\_damping is used to compute the z-axis element of damping of the body moment vector by a call to the CSU compute\_body\_damping\_forces\_and\_moments.

moment\_body\_damping[Z] = - yaw\_damping \* yaw\_rate;

See APPENDIX B for a complete source code listing.

# 3.1.71 VERTICAL\_RATE\_DAMPING\_GAIN

VERTICAL\_RATE\_DAMPING\_GAIN is a constant defining the vertical damping rate.

#### 3.1.71.1 Initialization

The constant VERTICAL\_RATE\_DAMPING\_GAIN is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define VERTICAL\_RATE\_DAMPING\_GAIN

aero\_data[70]

### 3.1.71.2 Usage

During real-time execution, this variable is not recomputed.

## 3.1.71.2.1 Algorithm

VERTICAL\_RATE\_DAMPING\_GAIN is used to compute the z-axis element of damping on the body force by a call to the CSU compute\_body\_damping\_forces\_and\_moments.

See APPENDIX B for a complete source code listing.

## 3.1.72 LATERAL\_VELOCITY\_DAMPING\_GAIN

LATERAL\_VELOCITY\_DAMPING\_GAIN is a constant defining the lateral velocity damping rate.

#### 3.1.72.1 Initialization

The constant LATERAL\_VELOCITY\_DAMPING\_GAIN is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define LATERAL\_VELOCITY\_DAMPING\_GAIN

aero\_data[71]

## 3.1.72.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.72.2.1 Algorithm

LATERAL\_VELOCITY\_DAMPING\_GAIN is used to compute the x-axis element of damping on the body force by a call to the CSU compute\_body\_damping\_forces\_and\_moments.

See APPENDIX B for a complete source code listing.

### 3.1.73 LIFT\_COEFF\_VIRTUAL\_WING

LIFT\_COEFF\_VIRTUAL\_WING is a constant defining the lift coefficient of the virtual wing.

#### 3.1.73.1 Initialization

The constant LIFT\_COEFF\_VIRTUAL\_WING is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define LIFT\_COEFF\_VIRTUAL\_WING

aero\_data[72]

## 3.1.73.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.73.2.1 Algorithm

LIFT\_COEFF\_VIRTUAL\_WING is used to initialize the lift coefficient of the virtual wing by a call to the CSU compute\_lift\_drag\_coefficients.

lift\_coefficient\_virtual\_wing = LIFT\_COEFF\_VIRTUAL\_WING;

The lift\_coefficient\_virtual\_wing is used to compute the lift force on the virtual wing by a call to the CSU compute\_lift\_drag\_forces.

lift\_coefficient\_virtual\_wing \* VIRTUAL\_WING\_AREA;

See APPENDIX B for a complete source code listing.

### 3.1.74 OSWALD\_EFFIC\_FACTOR

OSWALD\_EFFIC\_FACTOR is a constant defining the oswald efficiency factor.

#### 3.1.74.1 Initialization

The constant OSWALD\_EFFIC\_FACTOR is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define OSWALD\_EFFIC\_FACTOR

aero\_data[73]

## 3.1.74.2 Usage

During real-time execution, this variable is not recomputed.

# 3.1.74.2.1 Algorithm

OSWALD\_EFFIC\_FACTOR is used to initialize the oswald efficiency factor by a call to the CSU compute\_lift\_drag\_coefficients.

oswald\_efficiency\_factor = OSWALD\_EFFIC\_FACTOR;

See APPENDIX B for a complete source code listing.

# 3.1.75 INDUCED\_DRAG\_COEFF

INDUCED\_DRAG\_COEFF is a constant defining the induced drag coefficient.

#### 3.1.75.1 Initialization

The constant INDUCED\_DRAG\_COEFF is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.1. - Aerodynamics Data Array for a summary of the constant.

#define INDUCED\_DRAG\_COEFF

aero\_data[74]

#### 3.1.75.2 Usage

During real-time execution, this variable is not recomputed.

### 3.1.75.2.1 Algorithm

INDUCED\_DRAG\_COEFF is used to initialize the induced drag coefficient by a call to the CSU compute\_lift\_drag\_coefficients.

induced\_drag\_coefficient = INDUCED\_DRAG\_COEFF;

The induced\_drag\_coefficient is used to compute the total incompressible drag by a call to the CSU compute\_lift\_drag\_coefficients.

total\_incompressible\_drag\_coefficient = parasite\_drag\_coefficient + induced\_drag\_coefficient;

See APPENDIX B for a complete source code listing.

### 3.2 Aero init

This data array consists of initial values for positions of the control inputs, stabilator augmentation integrators, attitude control integrators, and hover augmentation integrators.

# 3.2.1 Cyclic\_pitch

Cyclic\_pitch is a variable defining the longitudinal position of the cyclic.

#### 3.2.1.1 Initialization

The variable cyclic\_pitch is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

```
cyclic_pitch = aero_init[ 0];
```

### 3.2.1.2 Usage

During real-time execution, this variable is recomputed. The normal range is from -1.0 to +1.0.

### 3.2.1.2.1 Algorithm

Cýclic\_pitch is used to compute the cyclic controller pitch input. The longitudinal position of the cyclic is set by a call to the CSU aerodyn\_set\_longitudinal\_stick to read the physical position.

```
void aerodyn_set_longitudinal_stick (val)
    REAL val;
{
    cyclic_pitch = -val;
}
```

The controller\_cyclic\_pitch is used to compute the pitching moment of the moment\_body\_main\_rotor vector during execution of CSU compute\_rotor\_forces\_and\_moments.

```
moment_body_main_rotor[X] =
- controller_cyclic_pitch *
MAIN_ROTOR_MAX_PITCH_MOMENT;
```

The cyclic\_pitch is used to compute the controller\_cyclic\_pitch in the CSC compute\_stab\_augmentation\_gains.

```
controller_cyclic_pitch = cyclic_pitch + stab_aug_pitch;
```

During execution of the simple flight model, the cyclic\_pitch is used to compute the lift\_factor in the CSC aerodyn\_simple\_simul.

During execution of the simple flight model, the cyclic\_pitch is used to compute the pitch element of the desired orientation vector and the torque required in the CSC aerodyn\_simple\_simul.

```
orient_vec[0] = H_KPR * - cyclic_pitch + hover_hold_additions[0];
```

During execution of the stealth flight model, the cyclic\_pitch is used to compute the desired rotation vector and the desired linear velocity vector in the CSC aerodyn\_stealth\_simul.

```
if (hover_hold_state == ON)
{ /* no linear velocity in X,Y, only pitch */
    desired_lin_vel[X] = desired_lin_vel[Y] = 0.0;
    desired_rot_vel[X] = -cyclic_pitch * cyclic_pitch * sign(cyclic_pitch);
    desired_rot_vel[Y] = 0.0;
}
else
    if (level_view)/* when not in pitch mode, level view */
      vehicle_set_orientation_matrix (level); /* identity matrix */
       vehicle_set_orientation (kinematics_get_heading());
       level view = FALSE;
    }
   desired_lin_vel[X] = cyclic_roll * cyclic_roll * sign (cyclic_roll)
       * H_SIDE_MUL;
   desired_lin_vel[Y] = cyclic_pitch * cyclic_pitch * sign (cyclic_pitch)
       * H FWD_MUL;
   desired\_rot\_vel[X] = desired\_rot\_vel[Y] = 0.0;
}
```

## 3.2.2 Cyclic\_roll

Cyclic\_roll is a variable defining the lateral position of the cyclic.

#### 3.2.2.1 Initialization

The variable cyclic\_roll is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

```
cyclic_roll = aero_init[ 1];
```

# 3.2.2.2 Usage

During real-time execution, this variable is recomputed. The normal range is from -1.0 to +1.0.

### 3.2.2.2.1 Algorithm

Cyclic\_roll is used to compute the cyclic controller roll input. The lateral position of the cyclic is set by a call to the CSU aerodyn\_set\_lateral\_stick to read the physical position.

```
void aerodyn_set_lateral_stick (val)
    REAL val;
{
    cyclic_roll = -val;
}
```

The controller\_cyclic\_roll is used to compute the rolling moment of the moment\_body\_main\_rotor vector during execution of CSU compute\_rotor\_forces\_and\_moments.

The cyclic\_roll is used to compute the controller\_cyclic\_roll in the CSC compute\_stab\_augmentation\_gains.

```
controller_cyclic_roll = cyclic_roll + stab_aug_roll;
```

During execution of the simple flight model, the cyclic\_roll is used to compute the roll element of the desired orientation vector and the torque required in the CSC aerodyn\_simple\_simul.

```
orient_vec[1] = H_KPR * cyclic_roll + hover_hold_additions[1];
```

During execution of the stealth flight model, the cyclic\_roll is used to compute the desired rotation vector and the desired linear velocity vector in the CSC aerodyn\_stealth\_simul.

```
if (hover_hold_state == ON)
{ /* no linear velocity in X,Y, only pitch */
    desired_lin_vel[X] = desired_lin_vel[Y] = 0.0;
    desired_rot_vel[X] = -cyclic_pitch * cyclic_pitch * sign(cyclic_pitch);
    desired rot vel[Y] = 0.0;
}
else
    if (level_view)/* when not in pitch mode, level view */
      vehicle_set_orientation_matrix (level); /* identity matrix */
       vehicle_set_orientation (kinematics_get_heading());
       level_view = FALSE;
    }
   desired_lin_vel[X] = cyclic_roll * cyclic_roll * sign (cyclic_roll)
      * H_SIDE_MUL;
   desired_lin_vel[Y] = cyclic_pitch * cyclic_pitch * sign (cyclic_pitch)
       * H_FWD_MUL;
   desired_rot_vel[X] = desired_rot_vel[Y] = 0.0;
}
```

#### 3.2.3 Collective

Collective is a variable defining the position of the collective.

#### 3.2.3.1 Initialization

The variable collective is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

```
if (selected_model != STEALTH_MODEL)
  collective = aero_init[ 2];
else
{
  collective = 0.5;
   allow_takeoff = TRUE;
}
```

#### 3.2.3.2 Usage

During real-time execution, this variable is recomputed. The normal range is from 0.0 to +1.0.

## 3.2.3.2.1 Algorithm

Collective is used to compute the collective controller input. The collective position of the cyclic is set by a call to the CSU aerodyn\_set\_collective to read the physical position.

```
void aerodyn_set_collective (val)
    REAL val;
{
    if (funny_little_kludge)
        collective = log10 (val * 9.0 + 1.0); /* or, how to make linear log */
    else
        collective = val;
}
```

Controller\_collective is used to compute rotor loads during execution of the CSU compute\_rotor\_loads.

Controller\_collective is used to compute main\_rotor\_thrust during execution of the CSU compute\_rotor\_forces\_and\_moments.

The collective is used to compute the controller\_collective in the CSC compute\_stab\_augmentation\_gains.

```
controller_collective = collective + stab_aug_climb;
```

During execution of the simple flight model, collective is used to compute a collective factor, which in turn is used to compute power in the CSC aerodyn\_simple\_simul.

```
coll_factor = max(0.0,collective - 0.3);
power = H_KP * coll_factor + hover_hold_additions[2];
power += gross_weight * collective/(H_K2+collective) * 1.25;
power = min (MAX_HELICOPTER_POWER, power);
power = max (0.0, power);
```

During execution of the stealth flight model, collective is adjusted for dead zone and for -1 to 1 range.

```
adj_collective = (collective - 0.5) * 2.0; /* change to -1 to 1 */
```

During execution of the stealth flight model, the adjusted collective input is limited during allow\_takeoff state.

```
if (allow_takeoff)
{
    if (adj_collective > 0.0)
    {
        allow_takeoff = FALSE;
    }
    else
    {
        adj_collective = 0.0;
    }
}
```

During execution of the stealth flight model, the adjusted collective input is used to compute the vertical component of the desired linear velocity vector.

```
desired_lin_vel[Z] = adj_collective * adj_collective *
'sign (adj_collective) * H_COLL_MUL;
```

See APPENDIX B for a complete source code listing.

#### 3.2.4 Pedal

Pedal is a variable defining the position of the pedals (yaw control).

#### 3.2.4.1 Initialization

The variable pedal is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

```
pedal = aero_init[ 3];
```

## 3.2.4.2 Usage

During real-time execution, this variable is recomputed. The normal range is from -1.0 to +1.0.

### 3.2.4.2.1 Algorithm

Pedal is used to compute the tail rotor controller input. The position of the pedals is set by a call to the CSU aerodyn\_set\_pedal to read the physical position.

```
void aerodyn_set_pedal (val)
    REAL val;
{
    pedal = val;
}
```

The pedal is used to compute the controller\_tail\_rotor in the CSC compute\_stab\_augmentation\_gains.

```
controller_tail_rotor = pedal + stab_aug_yaw;
```

During execution of the simple flight model, the pedal is used to compute the yaw element of the desired orientation vector in the CSC aerodyn\_simple\_simul.

```
orient_vec[2] = kinematics_get_yaw () + sign(pedal) * pedal * pedal * H_KY;
```

During execution of the stealth flight model, the pedal is used to compute vertical element of the desired rotation vector in the CSC aerodyn\_stealth\_simul.

```
desired_rot_vel[Z] = pedal * pedal * sign(pedal);
```

See APPENDIX B for a complete source code listing.

# 3.2.5 Stab\_aug\_pitch\_integrator

Stab\_aug\_pitch\_integrator is a variable defining the integrator for the stabilization augmentation pitch axis.

#### 3.2.5.1 Initialization

The variable stab\_aug\_pitch\_integrator is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

stab\_aug\_pitch\_integrator = aero\_init[ 4];

#### 3.2.5.2 Usage

During real-time execution, this variable is not used.

See APPENDIX B for a complete source code listing.

### 3.2.6 Stab\_aug\_roll\_integrator

Stab\_aug\_roll\_integrator is a variable defining the integrator for the stabilization augmentation roll axis.

#### 3.2.6.1 Initialization

The variable stab\_aug\_roll\_integrator is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

stab\_aug\_roll\_integrator = aero\_init[5];

### 3.2.6.2 Usage

During real-time execution, this variable is not used.

See APPENDIX B for a complete source code listing.

# 3.2.7 Stab\_aug\_yaw\_integrator

Stab\_aug\_yaw\_integrator is a variable defining the integrator for the stabilization augmentation yaw axis.

### 3.2.7.1 Initialization

The variable stab\_aug\_yaw\_integrator is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

stab\_aug\_yaw\_integrator = aero\_init[ 6];

## 3.2.7.2 Usage

During real-time execution, this variable is recomputed.

## 3.2.7.2.1 Algorithm

The stab\_aug\_yaw\_integrator is computed and limited, then used to compute the stab\_aug\_yaw and the tail rotor control input in the CSC compute\_stab\_augmentation\_gains.

```
if (hover_hold_state == ON)
   if ( !hover_hold_turned_on )
         hover_hold_turned_on = TRUE ;
         /* You should already be "hovering" (airspeed < 10 knots)
           for hover hold to show little visible swaying. */
         stab_aug_yaw_integrator = 0.0;
   stab_aug_yaw_integrator -=
         HOVER_AUG_YAW_I_GAIN * angular_velocity_vector[Z];
   if (stab_aug_yaw_integrator > 0.5) stab_aug_yaw_integrator = 0.5;
   if (stab_aug_yaw_integrator < -0.5) stab_aug_yaw_integrator = -0.5;
   stab_aug_yaw = - HOVER_AUG_YAW_P_GAIN *
         angular_velocity_vector[Z] + stab_aug_yaw_integrator;
   stab_aug_yaw = limiter (
         -MAX_STAB_AUG_YAW_CLIMB_CONTROL,
         stab_aug_yaw,
         MAX_STAB_AUG_YAW_CLIMB_CONTROL);
}
else
{
   stab_aug_yaw = 0.0;
}
controller_tail_rotor = pedal + stab_aug_yaw;
```

See APPENDIX B for a complete source code listing.

# 3.2.8 Stab\_aug\_climb\_integrator

Stab\_aug\_climb\_integrator is a variable defining the integrator for the stabilization augmentation climb axis.

## 3.2.8.1 Initialization

The variable stab\_aug\_climb\_integrator is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

```
stab_aug_climb_integrator = aero_init[ 7];
```

## 3.2.8.2 Usage

During real-time execution, this variable is recomputed.

# 3.2.8.2.1 Algorithm

The stab\_aug\_climb\_integrator is computed and limited, then used to compute the stab\_aug\_climb and the tail rotor control input in the CSC compute\_stab\_augmentation\_gains.

```
-MAX_STAB_AUG_YAW_CLIMB_CONTROL,
stab_aug_climb,
MAX_STAB_AUG_YAW_CLIMB_CONTROL);
....
} else
{
....
stab_aug_climb = 0.0;
....
} ....
controller_collective = collective + stab_aug_climb;
....
}
```

# 3.2.9 Attitude\_control\_pitch\_integrator

Attitude\_control\_pitch\_integrator is a variable defining the integrator for the attitude control pitch axis.

### 3.2.9.1 Initialization

The variable attitude\_control\_pitch\_integrator is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

```
attitude_control_pitch_integrator = aero_init[ 8];
```

# 3.2.9.2 Usage

During real-time execution, this variable is recomputed.

# 3.2.9.2.1 Algorithm

The attitude\_control\_pitch\_integrator is computed and limited, then used to compute the attitude\_control\_pitch\_command in the CSC set\_pitch\_attitude.

### 3.2.10 Attitude\_control\_roll\_integrator

Attitude\_control\_roll\_integrator is a variable defining the integrator for the attitude control roll axis.

#### 3.2.10.1 Initialization

The variable attitude\_control\_roll\_integrator is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

```
attitude_control_roll_integrator = aero_init[ 9];
```

### 3.2.10.2 Usage

During real-time execution, this variable is recomputed.

# 3.2.10.2.1 Algorithm

The attitude\_control\_roll\_integrator is computed and limited, then used to compute the attitude\_control\_roll\_command in the CSC set\_roll\_attitude.

## 3.2.11 Hover\_aug\_pitch\_integrator

Hover\_aug\_pitch\_integrator is a variable defining the integrator for the hover augmentation pitch axis.

#### 3.2.11.1 Initialization

The variable hover\_aug\_pitch\_integrator is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

```
hover_aug_pitch_integrator = aero_init[10];
```

#### 3.2.11.2 Usage

During real-time execution, this variable is recomputed.

# 3.2.11.2.1 Algorithm

The hover\_aug\_pitch\_integrator is computed and limited, then used to compute the hover\_aug\_pitch\_angle in the CSC compute\_stab\_augmentation\_gains.

```
if (hover_hold_state == ON)
     if (!hover_hold_turned_on )
           hover_hold_turned_on = TRUE;
           /* You should already be "hovering" (airspeed < 10 knots)
             for hover hold to show little visible swaying. */
           hover_aug_pitch_integrator =
                       HOVER_AUG_PITCH_RESET_VALUE;
     hover_aug_pitch_integrator +=
                 HOVER_AUG_PITCH_I_GAIN * velocity_vector[Y];
     hover_aug_pitch_integrator =
                 limiter(-0.2,hover_aug_pitch_integrator,0.2);
     hover_aug_pitch_angle = HOVER_AUG_PITCH_P_GAIN *
                 velocity_vector[Y]
                 + hover_aug_pitch_integrator;
     hover_aug_pitch_angle = limiter (-MAX_ATT_CTL_ANGLE,
                               hover_aug_pitch_angle,
                               MAX ATT_CTL_ANGLE);
     stab_aug_pitch = set_pitch_attitude (hover_aug_pitch_angle);
 }
 else
#ifdef notdef
     hover_aug_pitch_integrator = 0.0;
#endif
 }
```

# 3.2.12 Hover\_aug\_roll\_integrator

Hover\_aug\_roll\_integrator is a variable defining the integrator for the hover augmentation roll axis.

#### 3.2.12.1 Initialization

The variable hover\_aug\_roll\_integrator is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

```
hover_aug_roll_integrator = aero_init[11];
```

#### 3.2.12.2 Usage

During real-time execution, this variable is recomputed.

### 3.2.12.2.1 Algorithm

The hover\_aug\_roll\_integrator is computed and limited, then used to compute the hover\_aug\_roll\_angle in the CSC compute\_stab\_augmentation\_gains.

```
if (hover_hold_state == ON)
   if ( !hover_hold_turned_on )
         hover_hold_turned_on = TRUE ;
         /* You should already be "hovering" (airspeed < 10 knots)
           for hover hold to show little visible swaying. */
         hover_aug_pitch_integrator = 0.0;
         hover_aug_roll_integrator +=
               HOVER_AUG_ROLL_I_GAIN * velocity_vector[X];
         hover_aug_roll_integrator =
               limiter(-0.2,hover_aug_roll_integrator,0.2);
         hover_aug_roll_angle = HOVER_AUG_ROLL_P_GAIN *
               velocity_vector[X]
               + hover_aug_roll_integrator;
         hover_aug_roll_angle = limiter (-MAX_ATT_CTL_ANGLE,
                           hover_aug_roll_angle,
                           MAX_ATT_CTL_ANGLE);
         stab_aug_roll = set_roll_attitude (hover_aug_roll_angle);
```

```
else
{
....
#ifdef notdef
....
    hover_aug_roll_integrator = 0.0;
#endif
....
}
```

## 3.2.13 Hover\_aug\_pitch\_angle

Hover\_aug\_pitch\_angle is a variable defining the integrator for the stabilization augmentation climb axis.

#### 3.2.13.1 Initialization

The variable hover\_aug\_pitch\_angle is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

```
hover_aug_pitch_angle = aero_init[12];
```

### 3.2.13.2 Usage

During real-time execution, this variable is recomputed.

# 3.2.13.2.1 Algorithm

The hover\_aug\_pitch\_angle is computed from the hover\_aug\_pitch\_integrator and y-element of the velocity\_vector in the CSC compute\_stab\_augmentation\_gains.

```
if (hover_hold_state == ON)
     if (!hover_hold_turned_on )
           hover_hold_turned_on = TRUE ;
           /* You should already be "hovering" (airspeed < 10 knots)
             for hover hold to show little visible swaying. */
           hover_aug_pitch_integrator =
                       HOVER_AUG_PITCH_RESET_VALUE;
     hover_aug_pitch_integrator +=
                 HOVER_AUG_PITCH_I_GAIN * velocity_vector[Y];
     hover_aug_pitch_integrator =
                 limiter(-0.2,hover_aug_pitch_integrator,0.2);
     hover_aug_pitch_angle = HOVER_AUG_PITCH_P_GAIN *
                 velocity_vector[Y]
                 + hover_aug_pitch_integrator;
     hover_aug_pitch_angle = limiter (-MAX_ATT_CTL_ANGLE,
                               hover_aug_pitch_angle,
                               MAX_ATT_CTL_ANGLE);
     stab_aug_pitch = set_pitch_attitude (hover_aug_pitch_angle);
 }
  else
#ifdef notdef
     hover_aug_pitch_integrator = 0.0;
#endif
 }
```

# 3.2.14 Hover\_aug\_roll\_angle

Hover\_aug\_roll\_angle is a variable defining the integrator for the stabilization augmentation climb axis.

#### 3.2.14.1 Initialization

The variable hover\_aug\_roll\_angle is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.2. - Aerodynamics Initialization Data Array for a summary of the variable.

```
hover_aug_roll_angle = aero_init[13];
```

### 3.2.14.2 Usage

During real-time execution, this variable is recomputed.

### 3.2.14.2.1 Algorithm

The hover\_aug\_roll\_angle is computed from the hover\_aug\_roll\_integrator and the x-element of the velocity\_vector in the CSC compute\_stab\_augmentation\_gains.

```
if (hover_hold_state == ON)
   if (!hover_hold_turned_on )
         hover_hold_turned_on = TRUE ;
         /* You should already be "hovering" (airspeed < 10 knots)
           for hover hold to show little visible swaying. */
         hover_aug_pitch_integrator = 0.0;
         hover_aug_roll_integrator +=
               HOVER_AUG_ROLL_I_GAIN * velocity_vector[X];
         hover_aug_roll_integrator =
               limiter(-0.2,hover_aug_roll_integrator,0.2);
         hover_aug_roll_angle = HOVER_AUG_ROLL_P_GAIN *
               velocity_vector[X]
               + hover_aug_roll_integrator;
         hover_aug_roll_angle = limiter (-MAX_ATT_CTL_ANGLE,
                           hover aug_roll_angle,
                           MAX ATT_CTL_ANGLE);
         stab_aug_roll = set_roll_attitude (hover_aug_roll_angle);
```

### 3.3 Aero\_simple

This data array consists of characteristics and parameters describing the physical vehicle and its aerodynamic performance and control in the "simple" mode. The following constants are for the simplified dynamics model. The model is a modification of the aerodynamics model from the SAF. Global variables defined for the real aerodynamics are re-used here to allow overlap in generic routines for operations such as control inputs, init, etc.

## 3.3.1 MAX\_HELICOPTER\_POWER

MAX\_HELICOPTER\_POWER is a constant defining the maximum helicopter power.

#### 3.3.1.1 Initialization

The constant MAX\_HELICOPTER\_POWER is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.3. - Aerodynamics Simple Data Array for a summary of the constant.

#define MAX\_HELICOPTER\_POWER aero\_simple[ 0]

## 3.3.1.2 Usage

During real-time execution, this constant is not recomputed.

### 3.3.1.2.1 Algorithm

MAX\_HELICOPTER\_POWER is used to limit the maximum power of the vehicle during execution of the CSU aerodyn\_simple\_simul.

```
power = min (MAX_HELICOPTER_POWER, power);
```

See APPENDIX B for a complete source code listing.

### 3.3.2 MAX\_HH

MAX\_HH is a constant defining the maximum hover hold input.

### 3.3.2.1 Initialization

The constant MAX\_HH is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.3. - Aerodynamics Simple Data Array for a summary of the constant.

#define MAX\_HH

aero\_simple[ 1]

## 3.3.2.2 Usage

During real-time execution, this constant is not recomputed.

# 3.3.2.2.1 Algorithm

MAX\_HH is used to limit the hover hold control inputs of the vehicle during execution of the CSU aerodyn\_simple\_simul.

```
else
{
    hover_hold_additions[0] = 0;
    hover_hold_additions[1] = 0;
    hover_hold_additions[2] = 0;
}
```

### 3.3.3 H\_K1

H\_K1 is a constant defining the gain on position error.

#### 3.3.3.1 Initialization

The constant H\_K1 is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.3. - Aerodynamics Simple Data Array for a summary of the constant.

```
#define H_K1 aero_simple[ 2]
```

#### 3.3.3.2 Usage

During real-time execution, this constant is not recomputed.

## 3.3.3.2.1 Algorithm

H\_K1 is used to compute the angular velocity necessary to achieve the desired orientation in exactly one tick. (delta theta/ delta T). Then get the angular acceleration needed to get to that velocity in one frame for the vehicle during execution of the CSU aerodyn\_simple\_simul.

## 3.3.4 H\_K2

H\_K2 is a constant defining the gain on gravity term of power setting.

# 3.3.4.1 Initialization

The constant H\_K2 is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.3. - Aerodynamics Simple Data Array for a summary of the constant.

#define H\_K2

aero\_simple[3]

## 3.3.4.2 Usage

During real-time execution, this constant is not recomputed.

# 3.3.4.2.1 Algorithm

H\_K2 is used to compute power of the stealth vehicle during execution of the CSU aerodyn\_simple\_simul.

power += gross\_weight \* collective/(H\_K2+collective) \* 1.25;

See APPENDIX B for a complete source code listing.

# 3.3.5 H\_K7

H\_K7 is a constant defining the air drag coefficient.

## 3.3.5.1 Initialization

The constant H\_K7 is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.3. - Aerodynamics Simple Data Array for a summary of the constant.

#define H\_K7 aero\_simple[ 4]

### 3.3.5.2 Usage

During real-time execution, this constant is not recomputed.

### 3.3.5.2.1 Algorithm

H\_K7 is used to compute the drag force in the y-axis of the stealth vehicle during execution of the CSU aerodyn\_simple\_simul.

drag\_ptr[Y] = square(cur\_ptr[Y]) \* H\_K7;

See APPENDIX B for a complete source code listing.

### 3.3.6 H\_K8

H\_K8 is a constant defining the air drag coefficient.

#### 3.3.6.1 Initialization

The constant H\_K8 is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.3. - Aerodynamics Simple Data Array for a summary of the constant.

#define H\_K8 aero\_simple[ 5]

# 3.3.6.2 Usage

During real-time execution, this constant is not recomputed.

# 3.3.6.2.1 Algorithm

H\_K8 is used to drag force in the x- and z-axes of the vehicle during execution of the CSU aerodyn\_simple\_simul.

```
drag_ptr[X] = square(cur_ptr[X]) * H_K8;
drag_ptr[Z] = square(cur_ptr[Z]) * H_K8;
```

#### 3.3.7 H\_KP

H\_KP is a constant gain defining the power relationship with the collective input.

#### 3.3.7.1 Initialization

The constant H\_KP is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.3. - Aerodynamics Simple Data Array for a summary of the constant.

#define H\_KP aero\_simple[ 6]

### 3.3.7.2 Usage

During real-time execution, this constant is not recomputed.

# 3.3.7.2.1 Algorithm

H\_KP is used to compute power of the stealth vehicle during execution of the CSU aerodyn\_simple\_simul.

```
power = H_KP * coll_factor + hover_hold_additions[2];
```

See APPENDIX B for a complete source code listing.

# 3.3.8 H\_KPR

H\_KPR is a constant defining the pitch/roll constant, approximately pi/3.

#### 3.3.8.1 Initialization

The constant H\_KPR is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done

only once during CSCI initialization and is performed sequentially. See TABLE 5.1.3. - Aerodynamics Simple Data Array for a summary of the constant.

#define H_KPR	aero_simple[ 7]

## 3.3.8.2 Usage

During real-time execution, this constant is not recomputed.

### 3.3.8.2.1 Algorithm

H\_KPR is used to compute the torque required to achieve the desired orientation in pitch and roll of the vehicle during execution of the CSU aerodyn\_simple\_simul.

```
, orient_vec[0] = H_KPR * - cyclic_pitch + hover_hold_additions[0];
  orient_vec[1] = H_KPR * cyclic_roll + hover_hold_additions[1];
```

See APPENDIX B for a complete source code listing.

## 3.3.9 H\_KY

H\_KY is a constant defining the yaw constant, approximately pi/2.

#### 3.3.9.1 Initialization

The constant H\_KY is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.3. - Aerodynamics Simple Data Array for a summary of the constant.

#define H_KY	aero_simple[8]	-

# 3.3.9.2 Usage

During real-time execution, this constant is not recomputed.

### 3.3.9.2.1 Algorithm

H\_KY is used to compute the torque required to achieve the desired orientation in yaw of the vehicle during execution of the CSU aerodyn\_simple\_simul.

```
orient_vec[2] = kinematics_get_yaw () + sign(pedal) * pedal * pedal * H_KY;
```

See APPENDIX B for a complete source code listing.

### 3.3.10 H\_KH

H\_KH is a constant defining the hover hold gain on velocity term.

### 3.3.10.1 Initialization

The constant H\_KH is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.3. - Aerodynamics Simple Data Array for a summary of the constant.

#define H\_KH aero\_simple[ 9]

# 3.3.10.2 Usage

During real-time execution, this constant is not recomputed.

# 3.3.10.2.1 Algorithm

H\_KH is used to modify the computed limit on the velocity vector as an input to hover hold of the vehicle during execution of the CSU aerodyn\_simple\_simul.

```
H_KH,MAX_HH);
hover_hold_additions[1] = max(hover_hold_additions[1],-MAX_HH);
hover_hold_additions[2] = - velocity_vector[2] * H_KH * H_CHH;
}
else
{
hover_hold_additions[0] = 0;
hover_hold_additions[1] = 0;
hover_hold_additions[2] = 0;
}
```

#### 3.3.11 H\_CHH

H\_CHH is a constant defining the collective hover hold gain.

#### 3.3.11.1 Initialization

The constant H\_CHH is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.3. - Aerodynamics Simple Data Array for a summary of the constant.

```
#define H_CHH aero_simple[10]
```

### 3.3.11.2 Usage

During real-time execution, this constant is not recomputed.

# 3.3.11.2.1 Algorithm

H\_CHH is used to computed of the velocity vector as an input to hover hold of the vehicle during execution of the CSU aerodyn\_simple\_simul.

```
hover_hold_additions[1] = max(hover_hold_additions[1],-MAX_HH);
hover_hold_additions[2] = - velocity_vector[2] * H_KH * H_CHH;
}
else
{
   hover_hold_additions[0] = 0;
   hover_hold_additions[1] = 0;
   hover_hold_additions[2] = 0;
}
```

### 3.3.12 H\_CL

H\_CL is a constant defining the coefficient of lift.

## 3.3.12.1 Initialization

The constant H\_CL is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.3. - Aerodynamics Simple Data Array for a summary of the constant.

```
#define H_CL aero_simple[11]
```

# 3.3.12.2 Usage

During real-time execution, this constant is not recomputed.

# 3.3.12.2.1 Algorithm

H\_CL is used to compute the lift as a function of cyclic pitch and velocity of the vehicle during execution of the CSU aerodyn\_simple\_simul.

See APPENDIX B for a complete source code listing.

### 3.4 Aero\_stealth

This data array consists of characteristics and parameters describing the physical vehicle and its aerodynamic performance and control in the "stealth" mode. The following is for the simplified model incorporating the stealth dynamics. In this model, the cyclic changes the desired velocity.

# 3.4.1 H\_FWD\_MUL

H\_FWD\_MUL is a constant defining the slope of the cyclic pitch position squared versus forward velocity curve.

## 3.4.1.1 Initialization

The constant H\_FWD\_MUL is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.4. - Aerodynamics Stealth Data Array for a summary of the constant.

#define H\_FWD\_MUL aero\_stealth[ 0]

# 3.4.1.2 Usage

During real-time execution, this constant is not recomputed.

# 3.4.1.2.1 Algorithm

H\_FWD\_MUL is used to compute the desired linear velocity in the forward direction for the stealth vehicle during execution of the CSU aerodyn\_stealth\_simul.

desired\_lin\_vel[Y] = cyclic\_pitch \* cyclic\_pitch \* sign (cyclic\_pitch)
 \* H\_FWD\_MUL;

See APPENDIX B for a complete source code listing.

#### 3.4.2 H\_SIDE\_MUL

H\_SIDE\_MUL is a constant defining the slope of the cyclic roll position squared versus sideward velocity curve.

#### 3.4.2.1 Initialization

The constant H\_SIDE\_MUL is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.4. - Aerodynamics Stealth Data Array for a summary of the constant.

#define H\_SIDE\_MUL aero\_stealth[ 1]

### 3.4.2.2 Usage

During real-time execution, this constant is not recomputed.

### 3.4.2.2.1 Algorithm

H\_SIDE\_MUL is used to compute the desired linear velocity in the sideward direction for the stealth vehicle during execution of the CSU aerodyn\_stealth\_simul.

desired\_lin\_vel[X] = cyclic\_roll \* cyclic\_roll \* sign (cyclic\_roll)
 \* H\_SIDE\_MUL;

See APPENDIX B for a complete source code listing.

# 3.4.3 H\_COLL\_MUL

H\_COLL\_MUL is a constant defining the slope of the collective position squared versus vertical velocity curve.

## 3.4.3.1 Initialization

The constant H\_COLL\_MUL is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.4. - Aerodynamics Stealth Data Array for a summary of the constant.

#define H\_COLL\_MUL aero\_stealth[ 2]

### 3.4.3.2 Usage

During real-time execution, this constant is not recomputed.

## 3.4.3.2.1 Algorithm

H\_COLL\_MUL is used to compute the desired linear velocity in the vertical direction for the stealth vehicle during execution of the CSU aerodyn\_stealth\_simul.

desired\_lin\_vel[Z] = adj\_collective \* adj\_collective \*
 sign (adj\_collective) \* H\_COLL\_MUL;

See APPENDIX B for a complete source code listing.

### 3.4.4 MAX\_TORQUE

MAX\_TORQUE is a constant defining the maximum controller torque for the stealth vehicle.

#### 3.4.4.1 Initialization

The constant MAX\_TORQUE is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.4. - Aerodynamics Stealth Data Array for a summary of the constant.

#define MAX\_TORQUE aero\_stealth[3]

## 3.4.4.2 Usage

During real-time execution, this constant is not recomputed.

# 3.4.4.2.1 Algorithm

MAX\_TORQUE is used to limit the controller torque for the stealth vehicle during execution of the CSU aerodyn\_stealth\_simul.

```
moment_body[X] = min (MAX_TORQUE, moment_body[X]);
moment_body[Y] = min (MAX_TORQUE, moment_body[Y]);
moment_body[Z] = min (MAX_TORQUE, moment_body[Z]);

moment_body[X] = max (-MAX_TORQUE, moment_body[X]);
moment_body[Y] = max (-MAX_TORQUE, moment_body[Y]);
moment_body[Z] = max (-MAX_TORQUE, moment_body[Z]);
```

#### 3.4.5 MAX\_FORCE

MAX\_FORCE is a constant defining the maximum controller force for the stealth vehicle.

#### 3.4.5.1 Initialization

The constant MAX\_FORCE is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.4. - Aerodynamics Stealth Data Array for a summary of the constant.

```
#define MAX_FORCE aero_stealth[ 4]
```

### 3.4.5.2 Usage

During real-time execution, this constant is not recomputed.

## 3.4.5.2.1 Algorithm

MAX\_FORCE is used to limit the controller forces for the stealth vehicle during execution of the CSU aerodyn\_stealth\_simul.

```
force_body[X] = min (MAX_FORCE, force_body[X]);
force_body[Y] = min (MAX_FORCE, force_body[Y]);
force_body[Z] = min (MAX_FORCE, force_body[Z]);

force_body[X] = max (-MAX_FORCE, force_body[X]);
force_body[Y] = max (-MAX_FORCE, force_body[Y]);
force_body[Z] = max (-MAX_FORCE, force_body[Z]);
```

#### 3.4.6 MASS

MASS is a constant defining the

### 3.4.6.1 Initialization

The constant MASS is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.4. - Aerodynamics Stealth Data Array for a summary of the constant.

```
#define MASS aero_stealth[ 5]
```

# 3.4.6.2 Usage

During real-time execution, this constant is not recomputed.

# 3.4.6.2.1 Algorithm

MASS is used to compute the controller forces for the stealth vehicle during execution of the CSU aerodyn\_stealth\_simul.

```
force_body[X] = (desired_lin_vel[X] - velocity_vector[X])
          * MASS/DELTA_T;
force_body[Y] = (desired_lin_vel[Y] - velocity_vector[Y])
          * MASS/DELTA_T;
force_body[Z] = (desired_lin_vel[Z] - velocity_vector[Z])
          * MASS/DELTA_T;
```

See APPENDIX B for a complete source code listing.

### 3.4.7 INERTIA

INERTIA is a constant defining the inertia of the stealth vehicle.

### 3.4.7.1 Initialization

The constant INERTIA is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.4. - Aerodynamics Stealth Data Array for a summary of the constant.

#define INERTIA aero\_stealth[ 6]

## 3.4.7.2 Usage

During real-time execution, this constant is not recomputed.

# 3.4.7.2.1 Algorithm

INERTIA is used to compute the controller torque for the stealth vehicle during execution of the CSU aerodyn\_stealth\_simul.

See APPENDIX B for a complete source code listing.

# 3.4.8 DEAD\_ZONE

DEAD\_ZONE is a constant defining the dead zone of the controls.

# 3.4.8.1 Initialization

The constant DEAD\_ZONE is initialized during execution of the CSU aerodyn\_init, called by CSC rwa\_init. Execution of the CSU aerodyn\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.4. - Aerodynamics Stealth Data Array for a summary of the constant.

#define DEAD\_ZONE aero\_stealth[7]

### 3.4.8.2 Usage

During real-time execution, this constant is not recomputed nor used.

See APPENDIX B for a complete source code listing.

## 3.5 Engine\_data

This data array consists of characteristics and parameters describing the engine performance and control.

# 3.5.1 GOVERNOR\_ENGINE\_SPEED\_SETTING

The GOVERNOR\_ENGINE\_SPEED\_SETTING is a constant defining the maximum engine speed setting at 100 percent rpm.

## 3.5.1.1 Initialization

The constant GOVERNOR\_ENGINE\_SPEED\_SETTING is initialized during execution of the CSU engine\_init, called by CSC rwa\_init. Execution of the CSU engine\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.5. - Engine Data Array for a summary of the initialization variable.

#define GOVERNOR\_ENGINE\_SPEED\_SETTING engine\_data[ 0]

# 3.5.1.2 Usage

During real-time execution, this constant is not recomputed.

# 3.5.1.2.1 Algorithm

Engine power is computed as a function of GOVERNOR\_ENGINE\_SPEED\_SETTING.

engine\_power = gov\_p\_gain \*
 (GOVERNOR\_ENGINE\_SPEED\_SETTING - engine\_speed);

If the engine is working, GOVERNOR\_ENGINE\_SPEED\_SETTING is used to compute the integrator\_gain.

The constant GOVERNOR\_ENGINE\_SPEED\_SETTING is used to compute powertrain\_percent\_shaft\_speed.

```
powertrain_percent_shaft_speed = engine_speed /
GOVERNOR_ENGINE_SPEED_SETTING;
```

See APPENDIX C for a complete source code listing.

# 3.5.2 GOVERNOR\_P\_GAIN

The GOVERNOR\_P\_GAIN is a constant defining the maximum engine speed gain.

#### 3.5.2.1 Initialization

The constant GOVERNOR\_P\_GAIN is initialized during execution of the CSU engine\_init, called by CSC rwa\_init. Execution of the CSU engine\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.5. - Engine Data Array for a summary of the initialization variable.

#define	GOVERNOR	P	GAIN
" acitic	00 / ==		_

engine\_data[ 1]

## 3.5.2.2 Usage

During real-time execution, this constant is not recomputed.

## 3.5.2.2.1 Algorithm

Engine variable gov\_p\_gain is initialized during execution of CSC engine\_init to GOVERNOR\_P\_GAIN.

GOVERNOR\_P\_GAIN;

The variable gov\_p\_gain is used to compute engine\_power. The variable gov\_p\_gain is not recomputed during execution of CSU engine\_simul.

```
engine_power = gov_p_gain *
   (GOVERNOR_ENGINE_SPEED_SETTING - engine_speed);
```

See APPENDIX C for a complete source code listing.

# 3.5.3 GOVERNOR\_I\_GAIN

The GOVERNOR\_I\_GAIN is a constant defining the maximum engine speed gain rate of the integrator.

#### 3.5.3.1 Initialization

The constant GOVERNOR\_I\_GAIN is initialized during execution of the CSU engine\_init, called by CSC rwa\_init. Execution of the CSU engine\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.5. - Engine Data Array for a summary of the initialization variable.

engine\_data[2]

# 3.5.3.2 Usage

During real-time execution, this constant is not recomputed.

# 3.5.3.2.1 Algorithm

Engine variable gov\_i\_gain is initialized during execution of CSC engine\_init to GOVERNOR\_I\_GAIN.

```
gov_i_gain = GOVERNOR_I_GAIN;
```

If the engine\_status is WORKING, the variable gov\_i\_gain is used to compute integrator\_gain. The variable gov\_p\_gain is not recomputed during execution of CSU engine\_simul.

See APPENDIX C for a complete source code listing.

# 3.5.4 MAX\_ENGINE\_TORQUE

The MAX\_ENGINE\_TORQUE is a constant defining the maximum engine torque.

# 3.5.4.1 Initialization

The constant MAX\_ENGINE\_TORQUE is initialized during execution of the CSU engine\_init, called by CSC rwa\_init. Execution of the CSU engine\_init is normally done only once during CSCI initialization and is performed

Reference # W003092; 31 March 1993 VOLUME 1 of 3; Rev. 0.0

sequentially. See TABLE 5.1.5. - Engine Data Array for a summary of the initialization variable.

#define MAX\_ENGINE\_TORQUE engine\_data[ 3]

## 3.5.4.2 Usage

During real-time execution, this constant is not recomputed.

## 3.5.4.2.1 Algorithm

The constant MAX\_ENGINE\_TORQUE is used to compute the engine\_percent\_torque.

engine\_percent\_torque = engine\_drive\_torque /
 (MAX\_ENGINE\_TORQUE \* number\_of\_engines);

See APPENDIX C for a complete source code listing.

## 3.5.5 MIN\_ENGINE\_LOAD\_TORQUE

The MIN\_ENGINE\_LOAD\_TORQUE is a constant defining the minimum engine load torque.

#### 3.5.5.1 Initialization

The constant MIN\_ENGINE\_LOAD\_TORQUE is initialized during execution of the CSU engine\_init, called by CSC rwa\_init. Execution of the CSU engine\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.5. - Engine Data Array for a summary of the initialization variable.

#define MIN\_ENGINE\_LOAD\_TORQUE engine\_data[ 4]

# 3.5.5.2 Usage

During real-time execution, this constant is not recomputed.

### 3.5.5.2.1 Algorithm

The constant MIN\_ENGINE\_LOAD\_TORQUE is used to set the lower limit of engine\_load\_torque.

```
if (engine_load_torque < MIN_ENGINE_LOAD_TORQUE)
  engine_load_torque = MIN_ENGINE_LOAD_TORQUE;</pre>
```

See APPENDIX C for a complete source code listing.

# 3.5.6 MAX\_ENGINE\_PERCENT\_POWER

The MAX\_ENGINE\_PERCENT\_POWER is a constant defining the maximum engine percent of power available.

## 3.5.6.1 Initialization

The constant MAX\_ENGINE\_PERCENT\_POWER is initialized during execution of the CSU engine\_init, called by CSC rwa\_init. Execution of the CSU engine\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.5. - Engine Data Array for a summary of the initialization variable.

#define MAX\_ENGINE\_PERCENT\_POWER engine\_data[ 5]

# 3.5.6.2 Usage

During real-time execution, this constant is not recomputed.

# 3.5.6.2.1 Algorithm

If engine\_power is greater than MAX\_ENGINE\_PERCENT\_POWER, engine\_power is limited to the MAX\_ENGINE\_PERCENT\_POWER.

if (engine\_power > MAX\_ENGINE\_PERCENT\_POWER)
 engine\_power = MAX\_ENGINE\_PERCENT\_POWER;

See APPENDIX C for a complete source code listing.

### 3.5.7 ENGINE\_TORQUE\_INTERCEPT

The ENGINE\_TORQUE\_INTERCEPT is a constant defining the engine torque curve intercept for the linear engine torque equation.

#### 3.5.7.1 Initialization

The constant ENGINE\_TORQUE\_INTERCEPT is initialized during execution of the CSU engine\_init, called by CSC rwa\_init. Execution of the CSU engine\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.5. - Engine Data Array for a summary of the initialization variable.

#define ENGINE\_TORQUE\_INTERCEPT

engine\_data[6]

#### 3.5.7.2 Usage

During real-time execution, this constant is not recomputed.

### 3.5.7.2.1 Algorithm

The constant ENGINE\_TORQUE\_INTERCEPT is used to compute engine\_drive\_torque.

See APPENDIX C for a complete source code listing.

### 3.5.8 ENGINE\_TORQUE\_SLOPE

The ENGINE\_TORQUE\_SLOPE is a constant defining the engine torque curve slope for the linear engine torque equation.

#### 3.5.8.1 Initialization

The constant ENGINE\_TORQUE\_SLOPE is initialized during execution of the CSU engine\_init, called by CSC rwa\_init. Execution of the CSU engine\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.5. - Engine Data Array for a summary of the initialization variable.

#define ENGINE\_TORQUE\_SLOPE

engine\_data[7]

### 3.5.8.2 Usage

During real-time execution, this constant is not recomputed.

### 3.5.8.2.1 Algorithm

The constant ENGINE\_TORQUE\_SLOPE is used to compute engine\_drive\_torque.

See APPENDIX C for a complete source code listing.

## 3.5.9 NOSE\_GEARBOX\_RATIO

The NOSE\_GEARBOX\_RATIO is a constant defining the nose gearbox ratio.

#### 3.5.9.1 Initialization

The constant NOSE\_GEARBOX\_RATIO is initialized during execution of the CSU engine\_init, called by CSC rwa\_init. Execution of the CSU engine\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.5. - Engine Data Array for a summary of the initialization variable.

#define NOSE\_GEARBOX\_RATIO

engine\_data[8]

## 3.5.9.2 Usage

During real-time execution, this constant is not recomputed.

# 3.5.9.2.1 Algorithm

NOSE\_GEARBOX\_RATIO is used to compute turbine speed.

turbine\_speed = engine\_speed \* NOSE\_GEARBOX\_RATIO;

See APPENDIX C for a complete source code listing.

# 3.5.10 MAIN\_ROTOR\_GEAR\_RATIO

The MAIN\_ROTOR\_GEAR\_RATIO is a constant defining the main rotor gear ratio.

### 3.5.10.1 Initialization

The constant MAIN\_ROTOR\_GEAR\_RATIO is initialized during execution of the CSU engine\_init, called by CSC rwa\_init. Execution of the CSU engine\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.5. - Engine Data Array for a summary of the initialization variable.

#define MAIN\_ROTOR\_GEAR\_RATIO

engine\_data[9]

### 3.5.10.2 Usage

During real-time execution, this constant is not recomputed.

# 3.5.10.2.1 Algorithm

MAIN\_ROTOR\_GEAR\_RATIO is used to compute main\_rotor\_engine\_load.

MAIN\_ROTOR\_GEAR\_RATIO is used to compute main\_rotor\_shaft\_speed.

main\_rotor\_shaft\_speed = engine\_speed / MAIN\_ROTOR\_GEAR\_RATIO;

Reference # W003092; 31 March 1993 VOLUME 1 of 3; Rev. 0.0

MAIN\_ROTOR\_GEAR\_RATIO is used to compute main\_rotor\_drive\_torque.

See APPENDIX C for a complete source code listing.

# 3.5.11 TAIL\_ROTOR\_GEAR\_RATIO

The TAIL\_ROTOR\_GEAR\_RATIO is a constant defining the tail rotor gear ratio.

#### 3.5.11.1 Initialization

The constant TAIL\_ROTOR\_GEAR\_RATIO is initialized during execution of the CSU engine\_init, called by CSC rwa\_init. Execution of the CSU engine\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.5. - Engine Data Array for a summary of the initialization variable.

#define TAIL\_ROTOR\_GEAR\_RATIO

engine\_data[10]

## 3.5.11.2 Usage

During real-time execution, this constant is not recomputed.

# 3.5.11.2.1 Algorithm

TAIL\_ROTOR\_GEAR\_RATIO is used to compute tail\_rotor\_engine\_load.

tail\_rotor\_engine\_load = tail\_rotor\_load / TAIL\_ROTOR\_GEAR\_RATIO;

TAIL\_ROTOR\_GEAR\_RATIO is used to compute tail\_rotor\_shaft\_speed.

tail\_rotor\_shaft\_speed = engine\_speed / TAIL\_ROTOR\_GEAR\_RATIO;

#### 3.5.12 POWERTRAIN\_INERTIA

The POWERTRAIN\_INERTIA is a constant defining the powertrain inertia.

#### 3.5.12.1 Initialization

The constant POWERTRAIN\_INERTIA is initialized during execution of the CSU engine\_init, called by CSC rwa\_init. Execution of the CSU engine\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.5. - Engine Data Array for a summary of the initialization variable.

#define POWERTRAIN\_INERTIA

engine\_data[11]

#### 3.5.12.2 Usage

During real-time execution, this constant is not recomputed.

### 3.5.12.2.1 Algorithm

If engine\_status is WORKING, POWERTRAIN\_INERTIA is used to compute engine\_speed.

```
if (engine_status == WORKING)
    engine_speed += (engine_drive_torque - engine_load_torque)
    / POWERTRAIN_INERTIA;
```

See APPENDIX C for a complete source code listing.

## 3.5.13 MAX\_FUELFLOW

The MAX\_FUELFLOW is a constant defining the maximum engine fuel flow.

#### 3.5.13.1 Initialization

The constant MAX\_FUELFLOW is initialized during execution of the CSU engine\_init, called by CSC rwa\_init. Execution of the CSU engine\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.5. - Engine Data Array for a summary of the initialization variable.

#define	MAX	FUEL	FLOW
		_	

engine\_data[12]

### 3.5.13.2 Usage

During real-time execution, this constant is not recomputed.

# 3.5.13.2.1 Algorithm

MAX\_FUELFLOW is used to compute engine fuel\_flow.

fuel\_flow = engine\_percent\_torque \* MAX\_FUELFLOW;

See APPENDIX C for a complete source code listing.

## 3.6 Engine\_init\_data

This data array consists of initial values of the current engine state, performance, and control.

## 3.6.1 Engine\_power

The variable engine\_power is a computed variable defining the current state of engine power.

#### 3.6.1.1 Initialization

The variable engine\_power is initialized during execution of the CSU engine\_init, called by CSC rwa\_init. Execution of the CSU engine\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.6. - Engine Initialization Data Array for a summary of the initialization variable.

engine\_power = engine\_init\_data[0]

### 3.6.1.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine\_simul.

If the engine is out of fuel, the engine\_power is set to 0.0.

```
if (fuel_level_empty ()) /* Out of gas */
{
    engine_power = 0.0;
    engine_speed = 0.0;
}
```

The engine\_power is then used to compute the engine\_drive\_torque.

```
engine_drive_torque = engine_power * number_of_engines *
   (ENGINE_TORQUE_INTERCEPT - ENGINE_TORQUE_SLOPE *
   engine_speed);
```

See APPENDIX C for a complete source code listing.

## 3.6.2 Engine\_percent\_torque

The variable engine\_percent\_torque is a computed variable defining the current state of percent of engine torque.

#### 3.6.2.1 Initialization

The variable engine\_percent\_torque is initialized during execution of the CSU engine\_init, called by CSC rwa\_init. Execution of the CSU engine\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.6. - Engine Initialization Data Array for a summary of the initialization variable.

```
engine_percent_torque = engine_init_data[ 1]
```

#### 3.6.2.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine\_simul.

## 3.6.2.2.1 Algorithm

Percent of engine torque is computed as a function of engine\_drive\_torque and number of engines.

```
engine_percent_torque = engine_drive_torque /
    (MAX_ENGINE_TORQUE * number_of_engines);
```

The engine\_percent\_torque is used to compute the fuel\_flow.

```
fuel_flow = engine_percent_torque * MAX_FUELFLOW;
```

If the engine is starting, and the engine\_percent\_torque is less than .0101, the engine is not starting. limited to a minimum values. If the engine is starting, and the engine\_percent\_torque is greater than or equal to .0101, the engine\_percent\_torque is limited to a value of .01.

The engine\_percent\_torque is output to the torque meter display.

```
meter_torque_set (engine_percent_torque);
```

The engine\_percent\_torque is also used to compute engine and rotor sound.

See Appendix C for a complete source code listing.

# 3.6.3 Engine\_speed

The variable engine\_speed is a computed variable defining the current state of engine speed.

### 3.6.3.1 Initialization

The variable engine\_speed is initialized during execution of the CSU engine\_init, called by CSC rwa\_init. Execution of the CSU engine\_init is

normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.6. - Engine Initialization Data Array for a summary of the initialization variable.

```
engine_speed = engine_init_data[ 2];
```

## 3.6.3.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine\_simul.

## 3.6.3.2.1 Algorithm

If the engine is working, engine\_speed is computed as an incremental change each frame as a function of the difference between engine\_drive\_torque and engine\_load\_torque.

```
if (engine_status == WORKING)
  engine_speed += (engine_drive_torque - engine_load_torque)
  / POWERTRAIN_INERTIA;
```

If the engine is out of fuel, engine\_speed is set to 0.0.

```
if (fuel_level_empty ()) /* Out of gas */
{
    engine_power = 0.0;
    engine_speed = 0.0;
}
```

If the engine-speed is less than 0.0, the engine\_speed is limited to 0.0.

```
if (engine_speed < 0.0)
engine_speed = 0.0;
```

If an engine is broken, engine\_speed is set to 0.0.

```
void engine_break_engine ()
{
    engine_status = BROKEN;
    engine_speed = 0.0;
    number_of_engines = 1;
}
```

If the engine is working, engine\_speed is used to compute the integrator\_gain for the engine\_power computation.

The engine\_speed is used to compute the engine\_power.

```
engine_power = gov_p_gain *
(GOVERNOR_ENGINE_SPEED_SETTING - engine_speed);
```

The engine\_speed is used to compute the engine\_drive\_torque.

The engine\_speed is used to compute the turbine\_speed.

```
turbine_speed = engine_speed * NOSE_GEARBOX_RATIO;
```

The engine\_speed is used to compute the main\_rotor\_shaft\_speed.

```
main_rotor_shaft_speed = engine_speed /
MAIN_ROTOR_GEAR_RATIO;
```

The engine\_speed is used to compute the tail\_rotor\_shaft\_speed.

```
tail_rotor_shaft_speed = engine_speed / TAIL_ROTOR_GEAR_RATIO;
```

The engine\_speed is used to compute the powertrain\_percent\_shaft\_speed.

```
powertrain_percent_shaft_speed = engine_speed /
    GOVERNOR_ENGINE_SPEED_SETTING;
```

See Appendix C for a complete source code listing.

# 3.6.4 Integrator\_gain

The variable integrator\_gain is a computed variable defining the rate of change for engine\_power during each frame.

#### 3.6.4.1 Initialization

The variable integrator\_gain is initialized during execution of the CSU engine\_init, called by CSC rwa\_init. Execution of the CSU engine\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.6. - Engine Initialization Data Array for a summary of the initialization variable.

```
integrator_gain = engine_init_data[ 3];
```

### 3.6.4.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine\_simul, if the engine is working..

### 3.6.4.2.1 Algorithm

The integrator\_gain is computed as a function of engine\_speed. It is limited to a value between 0.5 and -0.5. If the engine is not working, the integrator\_gain is set to zero.

See Appendix C for a complete source code listing.

# 3.6.5 Last\_percent\_shaft\_speed

The variable last\_percent\_shaft\_speed is a computed variable defining the state of percent of powertrain shaft speed from the last frame.

#### 3.6.5.1 Initialization

The variable last\_percent\_shaft\_speed is initialized during execution of the CSU engine\_init, called by CSC rwa\_init. Execution of the CSU engine\_init

is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.6. - Engine Initialization Data Array for a summary of the initialization variable.

```
last_percent_shaft_speed = engine_init_data[ 4];
```

#### 3.6.5.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine\_simul.

### 3.6.5.2.1 Algorithm

Last\_percent\_shaft\_speed is computed by assignment of powertrain\_percent\_shaft\_speed after the computation of .powertrain\_percent\_shaft\_speed during the current frame. The change of powertrain shaft speed is compared to a delta. If the absolute change of the powertrain shaft speed is greater than the delta, the sound for the rotor is recomputed.

See Appendix C for a complete source code listing.

# 3.6.6 Last\_percent\_torque

The variable last\_percent\_torque is a computed variable defining the state of percent of engine torque from the previous frame.

#### 3.6.6.1 Initialization

The variable last\_percent\_torque is initialized during execution of the CSU engine\_init, called by CSC rwa\_init. Execution of the CSU engine\_init is

normally done only once during CSCI initialization and is performed sequentially. See TABLE 3.6. - Engine Initialization Data Array for a summary of the initialization variable.

```
last_percent_torque = engine_init_data[ 5];
```

### 3.6.6.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine\_simul. It is used to compute sound change of the engine.

### 3.6.6.2.1 Algorithm

Last\_percent\_torque is computed by assignment of engine\_percent\_torque after the computation of .engine\_percent\_torque during the current frame. The change of engine torque is compared to a delta. If the absolute change of the engine torque is greater than this delta, the sound for engine is recomputed.

See Appendix C for a complete source code listing.

# 3.6.7 Hours\_of\_flight

The variable hours\_of\_flight is a computed variable defining the current hours of flight.

#### 3.6.7.1 Initialization

The variable hours\_of\_flight is initialized during execution of the CSU engine\_init, called by CSC rwa\_init. Execution of the CSU engine\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.6. - Engine Initialization Data Array for a summary of the initialization variable.

hours\_of\_flight = engine\_init\_data[6];

### 3.6.7.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine\_simul.

## 3.6.7.2.1 Algorithm

Hours\_of\_flight is computed by incrementing the current hours\_of\_flight by the amount of time each frame of execution.

hours\_of\_flight += HOURS\_PER\_TICK;

See Appendix C for a complete source code listing.

## 3.7 Engine\_stat\_data

This data array consists of the initial values for flight time, engine status, number of engines, and powertrain damage status.

## 3.7.1 Minutes\_of\_flight

The variable minutes\_of\_flight is a computed variable defining the current minutes of flight.

#### 3.7.1.1 Initialization

The variable minutes\_of\_flight is initialized during execution of the CSU engine\_init, called by CSC rwa\_init. Execution of the CSU engine\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.7. - Engine Status Data Array for a summary of the status variable.

minutes\_of\_flight = engine\_stat\_data[0];

## 3.7.1.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine\_simul.

### 3.7.1.2.1 Algorithm

Minutes\_of\_flight is computed as a function of the hours\_of\_flight.

```
minutes_of_flight = (int) (hours_of_flight * 60);
```

If a failure has occurred to the engine subsystem, the minutes\_of\_flight is stored in the variable old\_minutes\_of\_flight for use in the next frame.

```
old_minutes_of_flight = minutes_of_flight;
```

See Appendix C for a complete source code listing.

## 3.7.2 Old\_minutes\_of\_flight

The variable minutes\_of\_flight is a computed variable defining the current minutes of flight.

#### 3.7.2.1 Initialization

The variable minutes\_of\_flight is initialized during execution of the CSU engine\_init, called by CSC rwa\_init. Execution of the CSU engine\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.7. - Engine Status Data Array for a summary of the status variable.

```
old_minutes_of_flight = engine_stat_data[ 1];
```

# 3.7.2.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine\_simul.

# 3.7.2.2.1 Algorithm

If a failure has occurred to the engine subsystem, the minutes\_of\_flight is stored in the variable old\_minutes\_of\_flight for use in the next frame.

```
old_minutes_of_flight = minutes_of_flight;
```

See Appendix C for a complete source code listing.

### 3.7.3 Engine\_status

The variable engine\_status is a computed variable defining the current state of the engine.

#### 3.7.3.1 Initialization

The variable engine\_status is initialized during execution of the CSU engine\_init, called by CSC rwa\_init. Execution of the CSU engine\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.7. - Engine Status Data Array for a summary of the status variable.

engine\_status = engine\_stat\_data[ 2];

# 3.7.3.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine\_simul.

## 3.7.3.2.1 Algorithm

If the engine\_status is WORKING, the integrator\_gain and engine\_power are computed.

If the engine\_status is WORKING, engine\_speed is computed.

```
if (engine_status == WORKING)
engine_speed += (engine_drive_torque - engine_load_torque)
/ POWERTRAIN_INERTIA;
```

If the engine\_status is BROKEN, sound is halted.

```
if (engine_status == BROKEN)/* crippled condition */
{
    sound_stop_cont_sound (SOUND_OF_STOP_ENGINE,
SOUND_OF_VARY_ENGINE);
    sound_stop_cont_sound (SOUND_OF_STOP_ROTOR,
SOUND_OF_VARY_ROTOR);
    fuel_flow *= 50.0; /* fuel leak */
}
```

If a failure has broken the engine subsystem, engine\_status is set to BROKEN.

```
void engine_break_engine ()
{
   engine_status = BROKEN;
   engine_speed = 0.0;
   number_of_engines = 1;
}
```

If the engine subsystem has been repaired, engine\_status is set to WORKING.

```
void engine_repair_engine ()
{
    engine_repair_engine_oil ();
    engine_status = WORKING;
    number_of_engines = 2;
}
```

See Appendix C for a complete source code listing.

# 3.7.4 Starting\_engine

The variable starting\_engine is a computed Boolean defining the current state of the engine in a starting mode.

### 3.7.4.1 Initialization

The variable starting\_engine is initialized during execution of the CSU engine\_init, called by CSC rwa\_init. Execution of the CSU engine\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.7. - Engine Status Data Array for a summary of the status variable.

```
starting_engine = engine_stat_data[ 3];
```

# 3.7.4.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine\_simul.

#### 3.7.4.2.1 Algorithm

If the engine is starting, engine\_percent\_torque is limited to .01 until it exceeds the delta, at which time the starting\_engine flag is set to FALSE.

See Appendix C for a complete source code listing.

## 3.7.5 Number\_of\_engines .

The variable starting\_engine is a computed Boolean defining the current state of the engine in a starting mode.

#### 3.7.5.1 Initialization

The variable starting\_engine is initialized during execution of the CSU engine\_init, called by CSC rwa\_init. Execution of the CSU engine\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.7. - Engine Status Data Array for a summary of the status variable.

```
number_of_engines = engine_stat_data[ 4];
```

#### 3.7.5.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine\_simul.

## 3.7.5.2.1 Algorithm

If the engine subsystem has been broken, the number\_of\_engines is reset to 1 by a call to CSC engine\_break\_engine.

```
void engine_break_engine ()
{
    engine_status = BROKEN;
    engine_speed = 0.0;
    number_of_engines = 1;
}
```

If the engine subsystem has been repaired, the number\_of\_engines is reset to 2 by a call to CSC engine\_repair\_engine.

```
void engine_repair_engine ()
{
    engine_repair_engine_oil ();
    engine_status = WORKING;
    number_of_engines = 2;
}
```

The number\_of\_engines is used to compute the engine\_drive\_torque.

```
engine_drive_torque = engine_power * number_of_engines *
     (ENGINE_TORQUE_INTERCEPT - ENGINE_TORQUE_SLOPE *
engine_speed);
```

The number\_of\_engines is used to compute the engine\_percent\_torque.

```
engine_percent_torque = engine_drive_torque /
(MAX_ENGINE_TORQUE * number_of_engines);
```

See Appendix C for a complete source code listing.

# 3.7.6 Engine\_is\_damaged

The variable engine\_is\_damaged is a computed Boolean defining the current state of the engine damage.

#### 3.7.6.1 Initialization

The variable engine\_is\_damaged is initialized during execution of the CSU engine\_init, called by CSC rwa\_init. Execution of the CSU engine\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.7. - Engine Status Data Array for a summary of the status variable.

```
engine_is_damaged = engine_stat_data[ 5];
```

### 3.7.6.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine\_simul.

## 3.7.6.2.1 Algorithm

If engine oil is damaged, engine\_is\_damaged is set to TRUE by a call to CSC engine\_damage\_engine\_oil.

```
void engine_damage_engine_oil ()
{
#if DO_CFAIL
    controls_start_failure_lamp_flashing (MASTER_CAUTION);
    controls_start_failure_lamp_flashing (ENGINE_FAILURE);
#endif
    engine_is_damaged = TRUE;
}
```

If engine oil is repaired, engine\_is\_damaged is set to FALSE by a call to CSC engine\_repair\_engine\_oil.

```
void engine_repair_engine_oil ()
{
#if DO_CFAIL
    controls_failure_lamp_off (ENGINE_FAILURE);
    engine_is_damaged = FALSE;
#endif
}
```

See Appendix C for a complete source code listing.

# 3.7.7 Transmission\_is\_damaged

The variable transmission\_is\_damaged is a computed Boolean defining the current state of the transmission damage.

## 3.7.7.1 Initialization

The variable transmission\_is\_damaged is initialized during execution of the CSU engine\_init, called by CSC rwa\_init. Execution of the CSU engine\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.7. - Engine Status Data Array for a summary of the status variable.

```
transmission_is_damaged = engine_stat_data[ 6];
```

# 3.7.7.2 Usage

During real-time execution, this variable is recomputed each frame of CSC engine\_simul.

# 3.7.7.2.1 Algorithm

If engine transmission filter is damaged, transmission\_is\_damaged is set to TRUE by a call to CSC engine\_damage\_transmission\_filter.

```
void engine_damage_transmission_filter ()
{
#if DO_SFAIL
    controls_start_failure_lamp_flashing (MASTER_CAUTION);
    controls_start_failure_lamp_flashing (TRANSMISSION_FAILURE);
    transmission_is_damaged = TRUE;
#endif
}
```

If engine transmission filter is repaired, transmission\_is\_damaged is set to FALSE by a call to CSC engine\_repair\_transmission\_filter.

```
void engine_repair_transmission_filter ()
{
#if DO_SFAIL
    controls_failure_lamp_off (TRANSMISSION_FAILURE);
    transmission_is_damaged = FALSE;
#endif
}
```

See Appendix C for a complete source code listing.

### 3.8 Kinemat\_data

This data array consists of kinematics constants and limits for the vehicle and its control.

### 3.8.1 GRAV\_CONSTANT

GRAV\_CONSTANT is a constant defining the gravitational constant.

#### 3.8.1.1 Initialization

The constant is initialized during execution of the CSU veh\_spec\_kinematics\_init, called by CSC rwa\_init. Execution of the CSU veh\_spec\_kinematics\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.8. - Kinematics Data Array for a summary of the constant.

```
#define GRAV_CONSTANT kinemat_data[ 0]
```

## 3.8.1.2 Usage

During real-time execution, this constant is not recomputed.

# 3.8.1.2.1 Algorithm

The constant is used to compute g\_force during execution of CSU veh\_spec\_kinematics\_simul.

```
g_force = gravity[Z] + (true_airspeed * ang_vel[X] / GRAV_CONSTANT);
```

See APPENDIX D for a complete source code listing.

## 3.8.2 SIN\_AOA\_LIMIT

SIN\_AOA\_LIMIT is a constant defining the sine of the angle\_of\_attack limit.

#### 3.8.2.1 Initialization

The constant is initialized during execution of the CSU veh\_spec\_kinematics\_init, called by CSC rwa\_init. Execution of the CSU veh\_spec\_kinematics\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.8. - Kinematics Data Array for a summary of the constant.

#define SIN\_AOA\_LIMIT kinemat\_data[ 1]

## 3.8.2.2 Usage

During real-time execution, this constant is not recomputed.

## 3.8.2.2.1 Algorithm

The constant is not used for any current computations.

See APPENDIX D for a complete source code listing.

# 3.8.3 COS\_AOA\_LIMIT

COS\_AOA\_LIMIT is a constant defining the cosine of the angle\_of\_attack limit.

### 3.8.3.1 Initialization

The constant is initialized during execution of the CSU veh\_spec\_kinematics\_init, called by CSC rwa\_init. Execution of the CSU veh\_spec\_kinematics\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.8. - Kinematics Data Array for a summary of the constant.

#define COS\_AOA\_LIMIT kinemat\_data[ 2]

# 3.8.3.2 Usage

During real-time execution, this constant is not recomputed.

### 3.8.3.2.1 Algorithm

The constant is not used for any current computations.

See APPENDIX D for a complete source code listing.

### 3.8.4 SIN\_YAW\_LIMIT

SIN\_YAW\_LIMIT is a constant defining the sine of the yaw limit.

#### 3.8.4.1 Initialization

The constant is initialized during execution of the CSU veh\_spec\_kinematics\_init, called by CSC rwa\_init. Execution of the CSU veh\_spec\_kinematics\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.8. - Kinematics Data Array for a summary of the constant.

#defii	ne SIN_YAW_LIMIT	kinemat_data[ 3]	
1			

### 3.8.4.2 Usage

During real-time execution, this constant is not recomputed.

# 3.8.4.2.1 Algorithm

The constant is not used for any current computations.

See APPENDIX D for a complete source code listing.

# 3.8.5 COS\_YAW\_LIMIT

COS\_YAW\_LIMIT is a constant defining the cosine of the yaw limit.

#### 3.8.5.1 Initialization

The constant is initialized during execution of the CSU veh\_spec\_kinematics\_init, called by CSC rwa\_init. Execution of the CSU veh\_spec\_kinematics\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.8. - Kinematics Data Array for a summary of the constant.

#define COS\_YAW\_LIMIT kinemat\_data[ 4]

#### 3.8.5.2 Usage

During real-time execution, this constant is not recomputed.

### 3.8.5.2.1 Algorithm

The constant is not used for any current computations.

See APPENDIX D for a complete source code listing.

## 3.8.6 DISPLAY\_SPEED\_LIMIT

DISPLAY\_SPEED\_LIMIT is a constant defining the lower limit of the displayed speed.

#### 3.8.6.1 Initialization

The constant is initialized during execution of the CSU veh\_spec\_kinematics\_init, called by CSC rwa\_init. Execution of the CSU veh\_spec\_kinematics\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.8. - Kinematics Data Array for a summary of the constant.

```
#define DISPLAY_SPEED_LIMIT kinemat_data[ 5]
```

## 3.8.6.2 Usage

During real-time execution, this constant is not recomputed.

# 3.8.6.2.1 Algorithm

The constant is used to control computation of the velocity\_pitch.

```
if (true_airspeed >= DISPLAY_SPEED_LIMIT)
    velocity_pitch = asin (vertical_speed);
else
    velocity_pitch = 0.0;
```

The constant is used to control computation of the normalized velocity vector.

```
REAL *kinematics_get_normalized_velocity_vector ()
{
   if (true_airspeed > DISPLAY_SPEED_LIMIT)
      return (norm_vel);
   else if (norm_vel[Y] >= 0.0)
      return (pos_unit_vel);
   else
      return (neg_unit_vel);
}
```

## 3.9 Kinemat\_init\_data

This data array consists of initial values for kinematics variables including velocity, angle-of-attack, pitch, altitude, heading, and g-force.

## 3,9.1 Pos\_unit\_vel

Pos\_unit\_vel is an array defining the positive unit velocity vector.

#### 3.9.1.1 Initialization

The array is initialized during execution of the CSU veh\_spec\_kinematics\_init, called by CSC rwa\_init. Execution of the CSU veh\_spec\_kinematics\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
pos_unit_vel[Y] = kinemat_init_data[ 1];
pos_unit_vel[Z] = kinemat_init_data[ 2];
```

## 3.9.1.2 Usage

During real-time execution, this array is not recomputed.

# 3.9.1.2.1 Algorithm

The array is returned as the normalized velocity vector under certain conditions.

```
REAL *kinematics_get_normalized_velocity_vector ()
{
   if (true_airspeed > DISPLAY_SPEED_LIMIT)
      return (norm_vel);
   else if (norm_vel[Y] >= 0.0)
      return (pos_unit_vel);
   else
      return (neg_unit_vel);
}
```

## 3.9.2 Neg\_unit\_vel

Neg\_unit\_vel is an array defining the negative unit velocity vector.

#### 3.9.2.1 Initialization

The array is initialized during execution of the CSU veh\_spec\_kinematics\_init, called by CSC rwa\_init. Execution of the CSU veh\_spec\_kinematics\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
neg_unit_vel[X] = kinemat_init_data[3];
neg_unit_vel[Y] = kinemat_init_data[4];
neg_unit_vel[Z] = kinemat_init_data[5];
```

### 3.9.2.2 Usage

During real-time execution, this array is not recomputed.

## 3.9.2.2.1 Algorithm

The array is returned as the normalized velocity vector under certain conditions.

```
REAL *kinematics_get_normalized_velocity_vector ()
{
   if (true_airspeed > DISPLAY_SPEED_LIMIT)
      return (norm_vel);
   else if (norm_vel[Y] >= 0.0)
      return (pos_unit_vel);
   else
      return (neg_unit_vel);
}
```

### 3.9.3 Sin\_aoa

Sin\_aoa is a variable defining the sine of the angle-of-attack.

#### 3.9.3.1 Initialization

The variable is initialized during execution of the CSU veh\_spec\_kinematics\_init, called by CSC rwa\_init. Execution of the CSU veh\_spec\_kinematics\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
sin_aoa = kinemat_init_data[ 6];
```

#### 3.9.3.2 Usage

During real-time execution, this variable is recomputed.

## 3.9.3.2.1 Algorithm

The value of sin\_aoa is set based on the value of the 'Z' component of the normalized velocity vector.

```
if (norm_vel[Z] - 1.0 > -E_NANO)
  sin_aoa = -1.0;
  cos_aoa = 0.0;
  \sin yaw = 0.0;
  cos_yaw = 1.0;
else if (norm_vel[Z] + 1.0 < E_NANO)
  \sin aoa = 1.0;
  \cos_a = 0.0;
  \sin_yaw = 0.0;
  cos_yaw = 1.0;
}
else
{
   sin_aoa = -norm_vel[Z];
   cos_aoa = sqrt (norm_vel[X] * norm_vel[X] + norm_vel[Y] *
                 norm_vel[Y]);
   sin_yaw = norm_vel[X] / cos_aoa;
   cos_yaw = norm_vel[Y] / cos_aoa;
```

Sin\_aoa is used to compute a component of the velocity\_to\_body matrix.

```
velocity_to_body[1][2] = -sin_aoa;
```

See APPENDIX D for a complete source code listing.

## 3.9.4 Cos\_aoa

Cos aoa is a variable defining the cosine of the angle-of-attack.

#### 3.9.4.1 Initialization

The variable is initialized during execution of the CSU veh\_spec\_kinematics\_init, called by CSC rwa\_init. Execution of the CSU veh\_spec\_kinematics\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
cos_aoa = kinemat_init_data[ 7];
```

## 3.9.4.2 Usage

During real-time execution, this variable is recomputed.

# 3.9.4.2.1 Algorithm

The value of cos\_aoa is set based on the value of the 'Z' component of the normalized velocity vector.

```
if (norm\_vel[Z] - 1.0 > -E\_NANO)
  sin_aoa = -1.0;
  cos_aoa = 0.0;
  \sin_yaw = 0.0;
 \cos_yaw = 1.0;
else if (norm_vel[Z] + 1.0 < E_NANO)
  sin_aoa = 1.0;
  cos_aoa = 0.0;
  \sin_yaw = 0.0;
  \cos_yaw = 1.0;
else
   sin_aoa = -norm_vel[Z];
   cos_aoa = sqrt (norm_vel[X] * norm_vel[X] + norm_vel[Y] *
                 norm_vel[Y]);
   sin_yaw = norm_vel[X] / cos_aoa;
   cos_yaw = norm_vel[Y] / cos_aoa;
```

Cos\_aoa is used to compute a components of the velocity\_to\_body matrix, roll and heading.

```
temp = cos_aoa;
velocity_to_body[1][0] = -velocity_to_body[0][1] * temp;
velocity_to_body[1][1] = velocity_to_body[0][0] * temp;
temp = sqrt (body_to_world[1][0] * body_to_world[1][0] +
    body_to_world[1][1] * body_to_world[1][1]);
if (temp < E_NANO)
  roll = 0.0;
  heading = 0.0;
}
else
{
    temp2 = (body\_to\_world[0][0] * body\_to\_world[1][1] -
           body_to_world[0][1] * body_to_world[1][0]) / temp;
    if (temp2 > 1.0) temp2 = 1.0;
    roll = acos (temp2);
  if (body_to_world[1][1] * body_to_world[2][0] -
       body_to_world[1][0] * body_to_world[2][1] < 0.0)
    roll = -roll;
  if (body_to_world[1][0] >= 0.0)
    heading = acos (body_to_world[1][1] / temp);
  else
    heading = acos (-body_to_world[1][1] / temp) + PI;
```

# 3.9.5 Sin\_yaw

Sin\_yaw is a variable defining the sine of the yaw angle.

#### 3.9.5.1 Initialization

The variable is initialized during execution of the CSU veh\_spec\_kinematics\_init, called by CSC rwa\_init. Execution of the CSU veh\_spec\_kinematics\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
sin_yaw = kinemat_init_data[ 8];
```

## 3.9.5.2 Usage

During real-time execution, this variable is recomputed.

# 3.9.5.2.1 Algorithm

The value of sin\_yaw is set based on the value of the 'Z' component of the normalized velocity vector.

```
if (norm_vel[Z] - 1.0 > -E_NANO)
   \sin_aoa = -1.0;
   \cos_{aoa} = 0.0;
   \sin_y aw = 0.0;
   cos_yaw = 1.0;
  else if (norm_vel[Z] + 1.0 < E_NANO)
    sin_aoa = 1.0;
    \cos_{aoa} = 0.0;
    \sin_yaw = 0.0;
    cos_yaw = 1.0;
 }
 else
     sin_aoa = -norm_vel[Z];
          cos_aoa = sqrt (norm_vel[X] * norm_vel[X] + norm_vel[Y] *
norm_vel[Y]);
     sin_yaw = norm_vel[X] / cos_aoa;
     cos_yaw = norm_vel[Y] / cos_aoa;
  }
```

The value of sin\_yaw is used to compute the value of a component of the velocity\_to\_body matrix.

```
velocity_to_body[0][1] = -sin_yaw;
```

See APPENDIX D for a complete source code listing.

#### 3.9.6 Cos\_yaw

Cos\_yaw is a variable defining the cosine of the yaw angle.

#### 3.9.6.1 Initialization

The variable is initialized during execution of the CSU veh\_spec\_kinematics\_init, called by CSC rwa\_init. Execution of the CSU veh\_spec\_kinematics\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
cos_yaw = kinemat_init_data[ 9];
```

### 3.9.6.2 Usage

During real-time execution, this variable is recomputed.

### 3.9.6.2.1 Algorithm

The value of cos\_yaw is set based on the value of the 'Z' component of the normalized velocity vector.

```
if (norm_vel[Z] - 1.0 > -E_NANO)
{
    sin_aoa = -1.0;
    cos_aoa = 0.0;
    sin_yaw = 0.0;
    cos_yaw = 1.0;
}
else if (norm_vel[Z] + 1.0 < E_NANO)
{
    sin_aoa = 1.0;
    cos_aoa = 0.0;
    sin_yaw = 0.0;
    cos_yaw = 1.0;
}</pre>
```

The value of cos\_yaw is used to compute the value of a component of the velocity\_to\_body matrix.

```
velocity_to_body[0][0] = cos_yaw;
```

See APPENDIX D for a complete source code listing.

#### 3.9.7 Altitude

Altitude is a variable defining the altitude above mean sea level, the database datum.

#### 3.9.7.1 Initialization

The variable is initialized during execution of the CSU veh\_spec\_kinematics\_init, called by CSC rwa\_init. Execution of the CSU veh\_spec\_kinematics\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
altitude = kinemat_init_data[10];
```

## 3.9.7.2 Usage

During real-time execution, this variable is recomputed.

# 3.9.7.2.1 Algorithm

The value of altitude is set by assignment of the 'Z' component of the position vector.

```
altitude = position[Z];
```

If the value of altitude is negative, the altitude is limited to 0.0.

```
if (altitude < 0.0)
altitude = 0.0;
```

See APPENDIX D for a complete source code listing.

## 3.9.8 Body\_pitch

Body\_pitch is a variable defining the angle of body pitch.

#### 3.9.8.1 Initialization

The variable is initialized during execution of the CSU veh\_spec\_kinematics\_init, called by CSC rwa\_init. Execution of the CSU veh\_spec\_kinematics\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
body_pitch = kinemat_init_data[11];
```

### 3.9.8.2 Usage

During real-time execution, this variable is recomputed.

# 3.9.8.2.1 Algorithm

The value of body\_pitch is computed as the arcsine of a component of the body\_to\_world matrix.

```
body_pitch = asin (body_to_world[1][2]);
```

External access to the body\_pitch is achieved through a call to the CSC kinematics\_get\_body\_pitch. The value return includes an offset constant that allows for the adjustment of the body\_pitch reference.

```
REAL kinematics_get_body_pitch ()
{
    return (body_pitch + body_pitch_offset);
}
```

## 3.9.9 Body\_pitch\_offset

Body\_pitch\_offset is a constant defining the offset angle of body pitch. This offset allows for the adjustment of the body\_pitch reference.

#### 3.9.9.1 Initialization

The constant is initialized during execution of the CSU veh\_spec\_kinematics\_init, called by CSC rwa\_init. Execution of the CSU veh\_spec\_kinematics\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
body_pitch_offset = kinemat_init_data[12];
```

## 3.9.9.2 Usage

During real-time execution, this constant is not recomputed.

See APPENDIX D for a complete source code listing.

# 3.9.10 Velocity\_pitch

Velocity\_pitch is a variable defining the cosine of the angle-of-attack.

#### 3.9.10.1 Initialization

The variable is initialized during execution of the CSU veh\_spec\_kinematics\_init, called by CSC rwa\_init. Execution of the CSU veh\_spec\_kinematics\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
velocity_pitch = kinemat_init_data[13];
```

#### 3.9.10.2 Usage

During real-time execution, this variable is recomputed.

### 3.9.10.2.1 Algorithm

The value of velocity\_pitch is computed as the arcsine of the vertical\_speed. If the true\_airspeed is small, then the velocity\_pitch is set to 0.0.

```
if (true_airspeed >= DISPLAY_SPEED_LIMIT)
    velocity_pitch = asin (vertical_speed);
else
    velocity_pitch = 0.0;
```

See APPENDIX D for a complete source code listing.

#### 3.9.11 Roll

Róll is a variable defining the roll angle of the vehicle.

#### 3.9.11.1 Initialization

The variable is initialized during execution of the CSU veh\_spec\_kinematics\_init, called by CSC rwa\_init. Execution of the CSU veh\_spec\_kinematics\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
roll = kinemat_init_data[14];
```

## 3.9.11.2 Usage

During real-time execution, this variable is recomputed.

# 3.9.11.2.1 Algorithm

The value of roll is computed from components of the body\_to\_world matrix.

```
temp = sqrt (body_to_world[1][0] * body_to_world[1][0] +
    body_to_world[1][1] * body_to_world[1][1]);
if (temp < E_NANO)
  roll = 0.0;
  heading = 0.0;
else
    temp2 = (body_to_world[0][0] * body_to_world[1][1] -
           body_to_world[0][1] * body_to_world[1][0]) / temp;
    if (temp2 > 1.0) temp2 = 1.0;
    roll = acos (temp2);
  if (body_to_world[1][1] * body_to_world[2][0] -
       body_{to}_{world[1][0]} * body_{to}_{world[2][1]} < 0.0)
     roll = -roll;
  if (body_to_world[1][0] \ge 0.0)
     heading = acos (body_to_world[1][1] / temp);
     heading = acos (-body_to_world[1][1] / temp) + PI;
```

# 3.9.12 Heading

Heading is a variable defining the heading angle the vehicle.

### 3.9.12.1 Initialization

The variable is initialized during execution of the CSU veh\_spec\_kinematics\_init, called by CSC rwa\_init. Execution of the CSU veh\_spec\_kinematics\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
heading = kinemat_init_data[15];
```

#### 3.9.12.2 Usage

During real-time execution, this variable is recomputed.

### 3.9.12.2.1 Algorithm

The value of heading is computed from components of the body\_to\_world matrix.

```
temp = sqrt (body_to_world[1][0] * body_to_world[1][0] +
    body_to_world[1][1] * body_to_world[1][1]);
if (temp < E_NANO)
  roll = 0.0;
  heading = 0.0;
else
{
    temp2 = (body_to_world[0][0] * body_to_world[1][1] -
           body_to_world[0][1] * body_to_world[1][0]) / temp;
    if (temp2 > 1.0) temp2 = 1.0;
    roll = acos (temp2);
  if (body_to_world[1][1] * body_to_world[2][0] -
       body_to_world[1][0] * body_to_world[2][1] < 0.0)
     roll = -roll;
  if (body_to_world[1][0] >= 0.0)
    heading = acos (body_to_world[1][1] / temp);
  else
    heading = acos (-body_to_world[1][1] / temp) + PI;
```

See APPENDIX D for a complete source code listing.

# 3.9.13 True\_airspeed

True\_airspeed is a variable defining the true airspeed of the vehicle.

## 3.9.13.1 Initialization

The variable is initialized during execution of the CSU veh\_spec\_kinematics\_init, called by CSC rwa\_init. Execution of the CSU veh\_spec\_kinematics\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
true_airspeed = kinemat_init_data[16];
```

## 3.9.13.2 Usage

During real-time execution, this variable is recomputed.

# 3.9.13.2.1 Algorithm

The value of true\_airspeed is computed from the velocity vector.

True\_airspeed is used to compute indicated\_airspeed.

```
indicated_airspeed = true_airspeed * sqrt (air_density (altitude) /
    air_density(0.0));
```

True\_airspeed is used to compute the normalized velocity vector.

```
if (true_airspeed < E_MILLI)
{
    norm_vel[X] = 0.0;
    norm_vel[Y] = 1.0;
    norm_vel[Z] = 0.0;
}
else
{
    norm_vel[X] = velocity[X] / true_airspeed;
    norm_vel[Y] = velocity[Y] / true_airspeed;
    norm_vel[Z] = velocity[Z] / true_airspeed;
}</pre>
```

True\_airspeed is used to compute g\_force.

```
g_force = gravity[Z] + (true_airspeed * ang_vel[X] / GRAV_CONSTANT);
,
```

True\_airspeed is used to control computation of velocity\_pitch.

```
if (true_airspeed >= DISPLAY_SPEED_LIMIT)
    velocity_pitch = asin (vertical_speed);
else
    velocity_pitch = 0.0;
```

When access externally to the normalized velocity vector is requested, true\_airspeed controls the value of the returned variable.

```
REAL *kinematics_get_normalized_velocity_vector ()
{
   if (true_airspeed > DISPLAY_SPEED_LIMIT)
      return (norm_vel);
   else if (norm_vel[Y] >= 0.0)
      return (pos_unit_vel);
   else
      return (neg_unit_vel);
}
```

# 3.9.14 Indicated\_airspeed

Indicated\_airspeed is a variable defining the indicated airspeed of the vehicle.

## 3.9.14.1 Initialization

The variable is initialized during execution of the CSU veh\_spec\_kinematics\_init, called by CSC rwa\_init. Execution of the CSU veh\_spec\_kinematics\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
indicated_airspeed = kinemat_init_data[17];
```

## 3.9.14.2 Usage

During real-time execution, this variable is recomputed.

# 3.9.14.2.1 Algorithm

The value of indicated\_airspeed is computed from the true\_airspeed and corrected for altitude.

See APPENDIX D for a complete source code listing.

# 3.9.15 G\_force

G\_force is a variable defining the "g" force exerted on the vehicle.

# 3.9.15.1 Initialization

The variable is initialized during execution of the CSU veh\_spec\_kinematics\_init, called by CSC rwa\_init. Execution of the CSU veh\_spec\_kinematics\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

g\_force = kinemat\_init\_data[18];

## 3.9.15.2 Usage

During real-time execution, this variable is recomputed.

# 3.9.15.2.1 Algorithm

The value of g\_force is set computed from the true\_airspeed and angular velocity.

g\_force = gravity[Z] + (true\_airspeed \* ang\_vel[X] / GRAV\_CONSTANT);

See APPENDIX D for a complete source code listing.

## 3,9.16 Vertical\_speed

Vertical\_speed is a variable defining the vertical speed of the vehicle.

## 3.9.16.1 Initialization

The variable is initialized during execution of the CSU veh\_spec\_kinematics\_init, called by CSC rwa\_init. Execution of the CSU veh\_spec\_kinematics\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

vertical\_speed = kinemat\_init\_data[19];

# 3.9.16.2 Usage

During real-time execution, this variable is recomputed.

# 3.9.16.2.1 Algorithm

The value of vertical\_speed is computed from the normalized velocity vector and gravity, and adjusted using the true\_airspeed.

```
vertical_speed = vec_dot_prod (norm_vel, gravity);

vertical_speed *= true_airspeed;
```

Vertical\_speed is used to compute velocity\_pitch.

```
if (true_airspeed >= DISPLAY_SPEED_LIMIT)
    velocity_pitch = asin (vertical_speed);
else
    velocity_pitch = 0.0;
```

See APPENDIX D for a complete source code listing.

## 3.9.17 Gravity

Gravity is a vector defining the gravity components of the vehicle.

#### 3.9.17.1 Initialization

The variable is initialized during execution of the CSU veh\_spec\_kinematics\_init, called by CSC rwa\_init. Execution of the CSU veh\_spec\_kinematics\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
gravity[X] = kinemat_init_data[20];
gravity[Y] = kinemat_init_data[21];
gravity[Z] = kinemat_init_data[22];
```

## 3.9.17.2 Usage

During real-time execution, this variable is recomputed.

# 3.9.17.2.1 Algorithm

The value of gravity is assigned from the components of the body\_to\_world matrix.

```
gravity[X] = body_to_world[0][2];
gravity[Y] = body_to_world[1][2];
gravity[Z] = body_to_world[2][2];
```

The 'Z' component of the gravity vector is used to compute g\_force.

```
g_force = gravity[Z] + (true_airspeed * ang_vel[X] / GRAV_CONSTANT);
```

The value of vertical\_speed is computed from the normalized velocity vector and gravity, and adjusted using the true\_airspeed.

```
vertical_speed = vec_dot_prod (norm_vel, gravity);
```

See APPENDIX D for a complete source code listing.

### 3.9.18 Norm\_vel

Norm\_vel is a vector defining the normalized velocity vector.

#### 3.9.18.1 Initialization

The variable is initialized during execution of the CSU veh\_spec\_kinematics\_init, called by CSC rwa\_init. Execution of the CSU veh\_spec\_kinematics\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.9. - Kinematics Initialization Data Array for a summary of the variable data.

```
norm_vel[X] = kinemat_init_data[23];

norm_vel[Y] = kinemat_init_data[24];

norm_vel[Z] = kinemat_init_data[25];
```

## 3.9.18.2 Usage

During real-time execution, this variable is recomputed.

### 3.9.18.2.1 Algorithm

The value of norm\_vel vector is computed from the true\_airspeed and velocity vector.

```
if (true_airspeed < E_MILLI)
{
    norm_vel[X] = 0.0;
    norm_vel[Y] = 1.0;
    norm_vel[Z] = 0.0;
}
else
{
    norm_vel[X] = velocity[X] / true_airspeed;
    norm_vel[Y] = velocity[Y] / true_airspeed;
    norm_vel[Z] = velocity[Z] / true_airspeed;
}</pre>
```

The norm\_vel is used to compute and control computation of the sin\_aoa, cos\_aoa, sin\_yaw, and cos\_yaw.

```
if (norm_vel[Z] - 1.0 > -E_NANO)
{
    sin_aoa = -1.0;
    cos_aoa = 0.0;
    sin_yaw = 0.0;
    cos_yaw = 1.0;
}
else if (norm_vel[Z] + 1.0 < E_NANO)
{
    sin_aoa = 1.0;
    cos_aoa = 0.0;
    sin_yaw = 0.0;
    cos_yaw = 1.0;
}</pre>
```

```
else
{
    sin_aoa = -norm_vel[Z];
        cos_aoa = sqrt (norm_vel[X] * norm_vel[X] + norm_vel[Y] *
    norm_vel[Y]);
    sin_yaw = norm_vel[X] / cos_aoa;
    cos_yaw = norm_vel[Y] / cos_aoa;
}
```

The value of vertical\_speed is computed from the normalized velocity vector and gravity, and adjusted using the true\_airspeed.

```
vertical_speed = vec_dot_prod (norm_vel, gravity);
```

See APPENDIX D for a complete source code listing.

# 3.10 Hellfr\_miss\_char

This data array consists of characteristics and parameters describing a Hellfire missile system and its performance constraints.

## 3.10.1 HELLFIRE\_ARM\_TIME

HELLFIRE\_ARM\_TIME is a constant defining the hellfire missile arm time delay before firing in ticks.

### 3.10.1.1 Initialization

The constant is initialized during execution of the CSU missile\_hellfire\_init, called by CSC weapons\_init. Execution of the CSU missile\_hellfire\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.10. - Hellfire Missile Characteristics Data Array for a summary of the constants data.

```
#define HELLFIRE_ARM_TIME hellfr_miss_char[ 0]
```

## 3.10.1.2 Usage

During real-time execution, this variable is not recomputed.

#### 3.10.1.2.1 Algorithm

HELLFIRE\_ARM\_TIME is used to control computation of the missile flyout path by a call to the CSC missile\_hellfire\_fly.

See APPENDIX G for a complete source code listing.

### 3.10.2 HELLFIRE\_BURNOUT\_TIME

HELLFIRE\_BURNOUT\_TIME is a constant defining the time of powered flight for hellfire missile in ticks.

#### 3.10.2.1 Initialization

The constant is initialized during execution of the CSU missile\_hellfire\_init, called by CSC weapons\_init. Execution of the CSU missile\_hellfire\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.10. - Hellfire Missile Characteristics Data Array for a summary of the constants data.

```
#define HELLFIRE_BURNOUT_TIME hellfr_miss_char[ 1]
```

#### 3.10.2.2 Usage

During real-time execution, this variable is not recomputed.

### 3.10.2.2.1 Algorithm

HELLFIRE\_BURNOUT\_TIME is used to control computation of the missile flyout speed by a call to the CSC missile\_hellfire\_fly.

See APPENDIX G for a complete source code listing.

# 3.10.3 HELLFIRE\_MAX\_FLIGHT\_TIME

HELLFIRE\_MAX\_FLIGHT\_TIME is a constant defining the maximum flight time for the hellfire missile assumed in ticks.

### 3.10.3.1 Initialization

The constant is initialized during execution of the CSU missile\_hellfire\_init, called by CSC weapons\_init. Execution of the CSU missile\_hellfire\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.10. - Hellfire Missile Characteristics Data Array for a summary of the constants data.

#define HELLFIRE\_MAX\_FLIGHT\_TIME hellfr\_miss\_char[ 2]

### 3.10.3.2 Usage

During real-time execution, this variable is not recomputed.

### 3.10.3.2.1 Algorithm

HELLFIRE\_MAX\_FLIGHT\_TIME is used to initialize the maximum flight time for an individual hellfire missile by a call to the CSU missile\_hellfire\_init.

```
mptr->max_flight_time = HELLFIRE_MAX_FLIGHT_TIME;
```

If not defined, the max\_flight\_time is set to the time-of-flight to the maximum range limit plus one by a call to the CSC missile\_hellfire\_fire.

```
#ifdef notdeff
  if( max_range_limit > 0.0 )
    mptr->max_flight_time =
       1.0 + missile_hellfire_calc_tof( max_range_limit );
#endif
```

See APPENDIX G for a complete source code listing.

### 3.10.4 SPEED\_0

SPEED\_0 is a constant defining the reference turn speed used to compute the ratio for the maximum turn angle.

#### 3.10.4.1 Initialization

The constant is initialized during execution of the CSU missile\_hellfire\_init, called by CSC weapons\_init. Execution of the CSU missile\_hellfire\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.10. - Hellfire Missile Characteristics Data Array for a summary of the constants data.

#define SPEED\_0 hellfr\_miss\_char[3]

### 3.10.4.2 Usage

During real-time execution, this variable is not recomputed.

### 3.10.4.2.1 Algorithm

SPEED\_0 is used to compute the maximum turn angle for an individual hellfire missile by a call to the CSC missile\_hellfire\_fly.

```
mptr->cos_max_turn[0] = cos (sqrt (mptr->speed / SPEED_0) * THETA_0);
```

See APPENDIX G for a complete source code listing.

#### 3.10.5 THETA\_0

THETA\_0 is a constant defining the reference maximum turn angle which is scaled for speed.

#### 3.10.5.1 Initialization

The constant is initialized during execution of the CSU missile\_hellfire\_init, called by CSC weapons\_init. Execution of the CSU missile\_hellfire\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.10. - Hellfire Missile Characteristics Data Array for a summary of the constants data.

		ļ
#define THETA_0	hellfr_miss_char[ 4]	

### 3.10.5.2 Usage

During real-time execution, this variable is not recomputed.

## 3.10.5.2.1 Algorithm

THETA\_0 is used to compute the maximum turn angle for an individual hellfire missile by a call to the CSC missile\_hellfire\_fly.

```
mptr->cos_max_turn[0] = cos (sqrt (mptr->speed / SPEED_0) * THETA_0);
```

### 3.10.6 SIN\_UNGUIDE

SIN\_UNGUIDE is a constant defining the sine of the delta pitch angle for an unguided hellfire missile

#### 3.10.6.1 Initialization

The constant is initialized during execution of the CSU missile\_hellfire\_init, called by CSC weapons\_init. Execution of the CSU missile\_hellfire\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.10. - Hellfire Missile Characteristics Data Array for a summary of the constants data.

#define SIN\_UNGUIDE hellfr\_miss\_char[ 5]

### 3.10.6.2 Usage

During real-time execution, this variable is not recomputed.

### 3.10.6.2.1 Algorithm

SIN\_UNGUIDE is used to compute the missile flyout path by a call to the CSC missile\_hellfire\_fly.

```
/*/
* If the missile is not armed, fly in a search trajectory; otherwise, fly
* in a targeted trajectory.
/*/
  if( max_range_limit > 0 &&
    kinematics_range_squared (veh_kinematics, mptr->location) >
   max_range_squared)
    missile target ground( mptr );
  else if (time < HELLFIRE_ARM_TIME)
     missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
                       SIN CLIMB, COS_CLIMB, SIN_LOCK,
                       COS_LOCK, COS_TERM, COS_LOSE);
 else
     missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                       COS_UNGUIDE, SIN_CLIMB, COS_CLIMB,
                       SIN_LOCK, COS_LOCK, COS_TERM, COS_LOSE);
```

#### 3.10.7 COS\_UNGUIDE

COS\_UNGUIDE is a constant defining the cosine of the delta pitch angle for an unguided hellfire missile

#### 3.10.7.1 Initialization

The constant is initialized during execution of the CSU missile\_hellfire\_init, called by CSC weapons\_init. Execution of the CSU missile\_hellfire\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.10. - Hellfire Missile Characteristics Data Array for a summary of the constants data.

#define COS\_UNGUIDE hellfr\_miss\_char[ 6]

### 3.10.7.2 Usage

During real-time execution, this variable is not recomputed.

### 3.10.7.2.1 Algorithm

COS\_UNGUIDE is used to compute the missile flyout path by a call to the CSC missile\_hellfire\_fly.

## 3.10.8 SIN\_CLIMB

SIN\_CLIMB is a constant defining the sine of the delta pitch angle for a climbing hellfire missile

#### 3.10.8.1 Initialization

The constant is initialized during execution of the CSU missile\_hellfire\_init, called by CSC weapons\_init. Execution of the CSU missile\_hellfire\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.10. - Hellfire Missile Characteristics Data Array for a summary of the constants data.

#define SIN_CLIMB	hellfr_miss_char[ 7]	<b>V</b> 50 (1)

### 3.10.8.2 Usage

During real-time execution, this variable is not recomputed.

## 3.10.8.2.1 Algorithm

SIN\_CLIMB is used to compute the missile flyout path by a call to the CSC missile\_hellfire\_fly.

#### 3.10.9 COS\_CLIMB

COS\_CLIMB is a constant defining the cosine of the delta pitch angle for a climbing hellfire missile

#### 3.10.9.1 Initialization

The constant is initialized during execution of the CSU missile\_hellfire\_init, called by CSC weapons\_init. Execution of the CSU missile\_hellfire\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.10. - Hellfire Missile Characteristics Data Array for a summary of the constants data.

#define COS\_CLIMB hellfr\_miss\_char[ 8]

### 3.10.9.2 Usage

Quring real-time execution, this variable is not recomputed.

### 3.10.9.2.1 Algorithm

COS\_CLIMB is used to compute the missile flyout path by a call to the CSC missile\_hellfire\_fly.

```
/*/

* If the missile is not armed, fly in a search trajectory; otherwise, fly

* in a targeted trajectory.

/*/

if( max_range_limit > 0 && kinematics_range_squared (veh_kinematics, mptr->location) > max_range_squared )

missile_target_ground( mptr );
else if (time < HELLFIRE_ARM_TIME)

missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE, SIN_CLIMB, COS_CLIMB, SIN_LOCK, COS_LOCK, COS_TERM, COS_LOSE);
else

missile_target_agm (mptr, target_location, SIN_UNGUIDE, COS_UNGUIDE, SIN_LOCK, COS_UNGUIDE, SIN_CLIMB, COS_CLIMB, SIN_LOCK, COS_UNGUIDE, SIN_LOCK, COS_UNGUIDE, SIN_LOCK, COS_LOCK, COS_TERM, COS_LOSE);
```

#### 3.10.10 SIN\_LOCK

SIN\_LOCK is a constant defining the sine of the lock cone angle for a lockedon hellfire missile

### 3.10.10.1 Initialization

The constant is initialized during execution of the CSU missile\_hellfire\_init, called by CSC weapons\_init. Execution of the CSU missile\_hellfire\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.10. - Hellfire Missile Characteristics Data Array for a summary of the constants data.

#define SIN\_LOCK

hellfr\_miss\_char[9]

#### 3.10.10.2 Usage

During real-time execution, this variable is not recomputed.

## 3.10.10.2.1 Algorithm

SIN\_LOCK is used to compute the missile flyout path by a call to the CSC missile\_hellfire\_fly.

```
/*/

* If the missile is not armed, fly in a search trajectory; otherwise, fly

* in a targeted trajectory.

/*/

if( max_range_limit > 0 && kinematics_range_squared (veh_kinematics, mptr->location) > max_range_squared )

missile_target_ground( mptr );
else if (time < HELLFIRE_ARM_TIME)

missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE, SIN_CLIMB, COS_CLIMB, SIN_LOCK, COS_LOCK, COS_LOCK, COS_LOCK, COS_LOSE);
else

missile_target_agm (mptr, target_location, SIN_UNGUIDE, COS_UNGUIDE, SIN_LOCK, COS_UNGUIDE, SIN_CLIMB, COS_CLIMB, SIN_LOCK, COS_UNGUIDE, SIN_LOCK, COS_LOCK, COS_LOCK, COS_TERM, COS_LOSE);
```

### 3.10.11 COS\_LOCK

COS\_LOCK is a constant defining the cosine of the lock cone angle for a locked-on hellfire missile

#### 3.10.11.1 Initialization

The constant is initialized during execution of the CSU missile\_hellfire\_init, called by CSC weapons\_init. Execution of the CSU missile\_hellfire\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.10. - Hellfire Missile Characteristics Data Array for a summary of the constants data.

#define COS\_LOCK

hellfr\_miss\_char[10]

### 3.10.11.2 Usage

During real-time execution, this variable is not recomputed.

### 3.10.11.2.1 Algorithm

COS\_LOCK is used to compute the missile flyout path by a call to the CSC missile hellfire\_fly.

## 3.10.12 COS\_TERM

COS\_TERM is a constant defining the cosine of the terminal pitch angle for a locked-on hellfire missile

## 3.10.12.1 Initialization

The constant is initialized during execution of the CSU missile\_hellfire\_init, called by CSC weapons\_init. Execution of the CSU missile\_hellfire\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.10. - Hellfire Missile Characteristics Data Array for a summary of the constants data.

#define COS\_TERM

hellfr\_miss\_char[11]

## 3.10.12.2 Usage

During real-time execution, this variable is not recomputed.

## 3.10.12.2.1 Algorithm

COS\_TERM is used to compute the missile flyout path by a call to the CSC missile\_hellfire\_fly.

```
/*/

* If the missile is not armed, fly in a search trajectory; otherwise, fly

* in a targeted trajectory.

/*/

if( max_range_limit > 0 && kinematics_range_squared (veh_kinematics, mptr->location) > max_range_squared )

missile_target_ground( mptr );
else if (time < HELLFIRE_ARM_TIME)

missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE, SIN_CLIMB, COS_CLIMB, SIN_LOCK, COS_LOCK, COS_LOCK, COS_LOCK, COS_LOSE);

else

missile_target_agm (mptr, target_location, SIN_UNGUIDE, COS_UNGUIDE, SIN_LOCK, COS_UNGUIDE, SIN_CLIMB, COS_CLIMB, SIN_LOCK, COS_UNGUIDE, SIN_LOCK, COS_LOCK, COS_LOCK, COS_LOSE);
```

#### 3.10.13 COS\_LOSE

COS\_LOSE is a constant defining the cosine of the pitch angle for a loss-of-lock-on hellfire missile

#### 3.10.13.1 Initialization

The constant is initialized during execution of the CSU missile\_hellfire\_init, called by CSC weapons\_init. Execution of the CSU missile\_hellfire\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.10. - Hellfire Missile Characteristics Data Array for a summary of the constants data.

#define COS\_LOSE hellfr\_miss\_char[12]

### 3.10.13.2 Usage

During real-time execution, this variable is not recomputed.

### 3.10.13.2.1 Algorithm

COS\_LOSE is used to compute the missile flyout path by a call to the CSC missile hellfire\_fly.

### 3.11 Hellfr\_miss\_poly\_deg

The hellfr\_miss\_poly\_deg array consists of values of the degree of each polynomial equation used to compute the time-of-flight, the burn speed, and the coast speed for the Hellfire missile.

### 3.11.1 HELLFIRE\_TOF\_DEG

HELLFIRE\_TOF\_DEG is a constant defining the polynomial degree for the hellfire missile time-of-flight coefficient data array. HELLFIRE\_TOF\_DEG is the first element of the hellfr\_miss\_poly\_deg array.

#### 3.11.1.1 Initialization

The constant is initialized during execution of the CSU missile\_hellfire\_init, called by CSC weapons\_init. Execution of the CSU missile\_hellfire\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.11. - Hellfire Missile Polynomial Degree Data Array for a summary of the constants data.

#define HELLFIRE_TOF_DEG	hellfr_miss_poly_deg[ 0]
,	

#### 3.11.1.2 Usage

During real-time execution, this variable is not recomputed. The maximum value for HELLFIRE\_TOF\_DEG is 9, especially, the declared size of the hellfire\_tof\_coeff array is 10.

## 3.11.1.2.1 Algorithm

HELLFIRE\_TOF\_DEG is used to compute the hellfire missile time of flight using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_hellfire\_calc\_tof. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

## 3.11.2 HELLFIRE\_BURN\_SPEED\_DEG

HELLFIRE\_BURN\_SPEED\_DEG is a constant defining the polynomial degree for the hellfire missile burn speed coefficient data array. HELLFIRE\_BURN\_SPEED\_DEG is the second element of the hellfr\_miss\_poly\_deg array.

### 3.11.2.1 Initialization

The constant is initialized during execution of the CSU missile\_hellfire\_init, called by CSC weapons\_init. Execution of the CSU missile\_hellfire\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.11. - Hellfire Missile Polynomial Degree Data Array for a summary of the constants data.

#define HELLFIRE\_BURN\_SPEED\_DEG hellfr\_miss\_poly\_deg[ 1]

# 3.11.2.2 Usage

During real-time execution, this variable is not recomputed. The maximum value for HELLFIRE\_BURN\_SPEED\_DEG is 9, especially, the declared size of the hellfire\_burn\_speed\_coeff array is 10.

## 3.11.2.2.1 Algorithm

HELLFIRE\_BURN\_SPEED\_DEG is used to compute the hellfire missile speed at launch using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_hellfire\_fire. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

HELLFIRE\_BURN\_SPEED\_DEG is used to compute the hellfire missile speed during powered flight [burn] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_hellfire\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

See APPENDIX G for a complete source code listing.

# 3.11.3 HELLFIRE\_COAST\_SPEED\_DEG

HELLFIRE\_COAST\_SPEED\_DEG is a constant defining the polynomial degree for the hellfire missile coast speed coefficient data array. HELLFIRE\_COAST\_SPEED\_DEG is the third element of the hellfr\_miss\_poly\_deg array.

#### 3.11.3.1 Initialization

The constant is initialized during execution of the CSU missile\_hellfire\_init, called by CSC weapons\_init. Execution of the CSU missile\_hellfire\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.11. - Hellfire Missile Polynomial Degree Data Array for a summary of the constants data.

#define HELLFIRE\_COAST\_SPEED\_DEG hellfr\_miss\_poly\_deg[ 2]

#### 3.11.3.2 Usage

During real-time execution, this variable is not recomputed. The maximum value for HELLFIRE\_COAST\_SPEED\_DEG is 9, especially, the declared size of the hellfire\_coast\_speed\_coeff array is 10.

## 3.11.3.2.1 Algorithm

HELLFIRE\_COAST\_SPEED\_DEG is used to compute the hellfire missile speed during unpowered flight [coast] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_hellfire\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

See APPENDIX G for a complete source code listing.

## 3.12 Hellfire\_tof\_coeff

The hellfire\_tof\_coeff array consists of the coefficients for a polynomial equation defining the Hellfire missile time-of-flight with respect to range in the form using the Newton-Raphson method.

#### 3.12.1 Initialization

The hellfire\_tof\_coeff array is initialized during execution of the CSU missile\_hellfire\_init, called by CSC weapons\_init. Execution of the CSU missile\_hellfire\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.12. - Hellfire Missile Time-Of-Flight Data Array for a summary of the constants data.

The array has a maximum size of 10 elements.

### 3.12.2 Usage

During real-time execution, this array is not recomputed. HELLFIRE\_TOF\_DEG determines the number of elements of the array to be used in the polynomial evaluation.

### 3.12.2.1 Algorithm

The hellfire\_tof\_coeff array is used to compute the hellfire missile time of flight using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_hellfire\_calc\_tof. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and range.

See APPENDIX G for a complete source code listing.

# 3.13 Hellfire\_burn\_speed\_coeff

The hellfire\_burn\_speed\_coeff array consists of the coefficients for a polynomial equation defining the Hellfire missile burn speed with respect to time in the form using the Newton-Raphson method.

#### 3.13.1 Initialization

The hellfire\_burn\_speed\_coeff array is initialized during execution of the CSU missile\_hellfire\_init, called by CSC weapons\_init. Execution of the CSU missile\_hellfire\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.13. - Hellfire Missile Burn Speed Data Array for a summary of the array data.

The array has a maximum size of 10 elements.

## 3.13.2 Usage

During real-time execution, this array is not recomputed. HELLFIRE\_BURN\_SPEED\_DEG determines the number of elements of the array to be used in the polynomial evaluation.

# 3.13.2.1 Algorithm

The hellfire\_burn\_speed\_coeff array is used to compute the hellfire missile speed at launch using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_hellfire\_fire. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

The hellfire\_burn\_speed\_coeff array is used to compute the hellfire missile speed during powered flight [burn] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_hellfire\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

### 3.14 Hellfire\_coast\_speed\_coeff

The hellfire\_coast\_speed\_coeff array consists of the coefficients for a polynomial equation defining the Hellfire missile coast speed with respect to time in the form using the Newton-Raphson method.

#### 3.14.1 Initialization

The hellfire\_coast\_speed\_coeff array is initialized during execution of the CSU missile\_hellfire\_init, called by CSC weapons\_init. Execution of the CSU missile\_hellfire\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.14. - Hellfire Missile Coast Speed Data Array for a summary of the constants data.

The array has a maximum size of 10 elements.

### 3.14.2 Usage

During real-time execution, this array is not recomputed. HELLFIRE\_COAST\_SPEED\_DEG determines the number of elements of the array to be used in the polynomial evaluation.

### 3.14.2.1 Algorithm

The hellfire\_coast\_speed\_coeff array is used to compute the hellfire missile speed during unpowered flight [coast] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_hellfire\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

See APPENDIX G for a complete source code listing.

### 3.15 Maverick\_miss\_char

The maverick\_miss\_char array consists of characteristics and parameters describing a Maverick missile system and its performance constraints.

## 3.15.1 MAVERICK\_ARM\_TIME

MAVERICK\_ARM\_TIME is a constant defining the maverick missile arm time delay before firing in ticks.

#### 3.15.1.1 Initialization

The constant is initialized during execution of the CSU missile\_maverick\_init, called by CSC weapons\_init. Execution of the CSU missile\_maverick\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

#define MAVERICK\_ARM\_TIME maverick\_miss\_char[ 0]

### 3.15.1.2 Usage

During real-time execution, this constant is not recomputed.

## 3.15.1.2.1 Algorithm

MAVERICK\_ARM\_TIME is used to control computation of the missile flyout path by a call to the CSC missile\_maverick\_fly.

```
/*/

* Find the target point to which the missile is to fly. The missile ignores

* any targets until it is armed.

/*/

if (time < MAVERICK_ARM_TIME)

missile_target_agm (mptr, NULL, SIN_UNGUIDE,

COS_UNGUIDE, SIN_CLIMB,

COS_CLIMB, SIN_LOCK,

COS_LOCK, COS_TERM, COS_LOSE);

else

{

TObjectP object = mvptr -> object_being_tracked;
```

```
Try to find a target. If one is found, fly towards it in the
    proper trajectory, otherwise, fly in a search trajectory.
/*/
    if (object != NO_OBJECT)
        VECTOR target_location;
        GetLocationOfTObject (object, target_location);
       mvptr->target_vehicle_id = object -> var.vehicleID;
        missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                             COS_UNGUIDE, SIN_CLIMB, COS_CLIMB,
                               SIN LOCK, COS_LOCK,
                               COS_TERM, COS_LOSE);
    }
    else
        mvptr->target_vehicle_id.vehicle = vehicleIrrelevant;
         if (TrackAcquire (mvptr -> sensor_id, veh_list, mptr -> location,
                      mptr -> orientation[1]) < 0)
            printf ("missile_maverick_fly: TrackAcquire: %s\n",
                   TrackErrString ());
        missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
                               SIN_CLIMB, COS_CLIMB, SIN_LOCK,
                               COS_LOCK, COS_TERM, COS_LOSE);
    }
```

See APPENDIX I for a complete source code listing.

# 3.15.2 MAVERICK\_BURNOUT\_TIME

MAVERICK\_BURNOUT\_TIME is a constant defining the time of powered flight for maverick missile in ticks.

## 3.15.2.1 Initialization

The constant is initialized during execution of the CSU missile\_maverick\_init, called by CSC weapons\_init. Execution of the CSU missile\_maverick\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

#define MAVERICK\_BURNOUT\_TIME maverick\_miss\_char[ 1]

#### 3.15.2.2 Usage

During real-time execution, this constant is not recomputed.

### 3.15.2.2.1 Algorithm

MAVERICK\_BURNOUT\_TIME is used to control computation of the missile flyout speed by a call to the CSC missile\_maverick\_fly.

See APPENDIX I for a complete source code listing.

# 3.15.3 MAVERICK\_MAX\_FLIGHT\_TIME

MAVERICK\_MAX\_FLIGHT\_TIME is a constant defining the maximum flight time for the maverick missile assumed in ticks.

### 3.15.3.1 Initialization

The constant is initialized during execution of the CSU missile\_maverick\_init, called by CSC weapons\_init. Execution of the CSU missile\_maverick\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

#define MAVERICK\_MAX\_FLIGHT\_TIME maverick\_miss\_char[ 2]

### 3.15.3.2 Usage

During real-time execution, this constant is not recomputed.

### 3.15.3.2.1 Algorithm

MAVERICK\_MAX\_FLIGHT\_TIME is used to initialize the maximum flight time for an individual maverick missile by a call to the CSU missile\_maverick\_init.

```
mptr->max_flight_time = MAVERICK_MAX_FLIGHT_TIME;
```

The max\_flight\_time for the each missile is set to the maximum flight time for an individual maverick missile by a call to the CSU missile maverick\_init.

See APPENDIX I for a complete source code listing.

# 3.15.4 MAVERICK\_LOCK\_THRESHOLD

MAVERICK\_LOCK\_THRESHOLD is a constant defining the cosine squared of the lock threshold angle for the maverick missile.

### 3.15.4.1 Initialization

The constant is initialized during execution of the CSU missile\_maverick\_init, called by CSC weapons\_init. Execution of the CSU missile\_maverick\_init is normally done only once during CSCI initialization

and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

```
#define MAVERICK_LOCK_THRESHOLD maverick_miss_char[ 3]
```

## 3.15.4.2 Usage

During real-time execution, this constant is not recomputed.

## 3.15.4.2.1 Algorithm

MAVERICK\_LOCK\_THRESHOLD is used to initialize the maximum cone threshold angle for the maverick missile by a call to the CSU missile\_maverick\_init.

```
maverick_cone_threshold = MAVERICK_LOCK_THRESHOLD;
```

The maverick\_cone\_of\_threshold and detectibility are computed by a call to the CSC missile\_maverick\_detectibility.

```
detectibility = sign (dotProduct) * dotProduct * dotProduct /
         vec_dot_prod (to_target, to_target);
/* if the object is outside the detection cone of the sensor,
 * return a detectibility of 0.
*/
 if ((mvptr = missile_maverick_get_missile_from_sensor_id (sensor_id))
    != NULL)
{
    switch (mvptr -> mptr.state)
      case MAVERICK_READY:
       maverick_cone_threshold = MAVERICK_LOCK_THRESHOLD;
      break:
      case MAVERICK_FLYING:
       maverick_cone_threshold = MAVERICK_HOLD_THRESHOLD;
      break:
      case MAVERICK_FREE:
     default:
       printf ("MaverickDetectibility: Maverick not READY or FLYING\n");
```

See APPENDIX I for a complete source code listing.

## 3.15.5 MAVERICK\_HOLD\_THRESHOLD

MAVERICK\_HOLD\_THRESHOLD is a constant defining the cosine squared of the hold threshold angle for the maverick missile.

### 3.15.5.1 Initialization

The constant is initialized during execution of the CSU missile\_maverick\_init, called by CSC weapons\_init. Execution of the CSU missile\_maverick\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

```
#define MAVERICK_HOLD_THRESHOLD maverick_miss_char[ 4]
```

#### 3.15.5.2 Usage

During real-time execution, this constant is not recomputed.

# 3.15.5.2.1 Algorithm

MAVERICK\_HOLD\_THRESHOLD is used to compute the maverick\_cone\_of\_threshold and detectibility by a call to the CSC missile\_maverick\_detectibility.

```
detectibility = sign (dotProduct) * dotProduct * dotProduct /
         vec_dot_prod (to_target, to_target);
/* if the object is outside the detection cone of the sensor,
* return a detectibility of 0.
*/
 if ((mvptr = missile_maverick_get_missile_from_sensor_id (sensor_id))
    != NULL)
    switch (mvptr -> mptr.state)
     case MAVERICK_READY:
       maverick_cone_threshold = MAVERICK_LOCK_THRESHOLD;
     case MAVERICK_FLYING:
       maverick_cone_threshold = MAVERICK_HOLD_THRESHOLD;
     case MAVERICK_FREE:
     default:
      printf ("MaverickDetectibility: Maverick not READY or FLYING\n");
       maverick_cone_threshold = MAVERICK_LOCK_THRESHOLD;
      break;
    if (detectibility < maverick_cone_threshold)</pre>
    detectibility = 0.0;
}
else
    printf ("MaverickDetectibility: no missile for sensorID %d\n",
          sensor_id);
return (detectibility);
```

See APPENDIX I for a complete source code listing.

### 3.15.6 SPEED\_0

SPEED\_0 is a constant defining the reference turn speed used to compute the ratio for the maximum turn angle.

## 3.15.6.1 Initialization

The constant is initialized during execution of the CSU missile\_maverick\_init, called by CSC weapons\_init. Execution of the CSU missile\_maverick\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

#define SPEED\_0 maverick\_miss\_char[ 5]

## 3.15.6.2 Usage

During real-time execution, this constant is not recomputed.

# 3.15.6.2.1 Algorithm

SPEED\_0 is used to compute the maximum turn angle for an individual maverick missile by a call to the CSC missile\_maverick\_fly.

```
/*/

* Note that this is a temporary method of finding turn angle.

/*/

mptr->cos_max_turn[0] = cos (sqrt (mptr->speed / (SPEED_0 + mptr->init_speed)) * THETA_0);
```

See APPENDIX I for a complete source code listing.

## 3.15.7 THETA\_0

THETA\_0 is a constant defining the reference maximum turn angle which is scaled for speed.

## 3.15.7.1 Initialization

The constant is initialized during execution of the CSU missile\_maverick\_init, called by CSC weapons\_init. Execution of the CSU missile\_maverick\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

#define THETA\_0

maverick\_miss\_char[6]

### 3.15.7.2 Usage

During real-time execution, this constant is not recomputed.

## 3.15.7.2.1 Algorithm

THETA\_0 is used to compute the maximum turn angle for an individual maverick missile by a call to the CSC missile\_maverick\_fly.

```
/*/

* Note that this is a temporary method of finding turn angle.

/*/

mptr->cos_max_turn[0] = cos (sqrt (mptr->speed / (SPEED_0 + mptr->init_speed)) * THETA_0);
```

See APPENDIX I for a complete source code listing.

## 3.15.8 SIN\_UNGUIDE

SIN\_UNGUIDE is a constant defining the sine of the delta pitch angle for an unguided maverick missile

### 3.15.8.1 Initialization

The constant is initialized during execution of the CSU missile\_maverick\_init, called by CSC weapons\_init. Execution of the CSU missile\_maverick\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

#define SIN\_UNGUIDE maverick\_miss\_char[ 7]

## 3.15.8.2 Usage

During real-time execution, this constant is not recomputed.

## 3.15.8.2.1 Algorithm

SIN\_UNGUIDE is used to compute the missile flyout path by a call to the CSC missile\_maverick\_fly.

```
/*/
* Find the target point to which the missile is to fly. The missile ignores
* any targets until it is armed.
  if (time < MAVERICK_ARM_TIME)
     missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
                        SIN_CLIMB, COS_CLIMB, SIN_LOCK,
                        COS_LOCK, COS_TERM, COS_LOSE);
 else
 {
     TObjectP object = mvptr -> object_being_tracked;
/*/
    Try to find a target. If one is found, fly towards it in the
    proper trajectory, otherwise, fly in a search trajectory.
/*/
    if (object != NO_OBJECT)
        VECTOR target_location;
        GetLocationOfTObject (object, target_location);
       mvptr->target_vehicle_id = object -> var.vehicleID;
        missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                              COS_UNGUIDE, SIN_CLIMB, COS_CLIMB,
                              SIN_LOCK, COS_LOCK, COS_TERM,
                               COS_LOSE);
    }
    else
        mvptr->target_vehicle_id.vehicle = vehicleIrrelevant;
        if (TrackAcquire (mvptr -> sensor_id, veh_list, mptr -> location,
                      mptr -> orientation[1]) < 0)
            printf ("missile_maverick_fly: TrackAcquire: %s\n",
                  TrackErrString ());
        missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
            SIN_CLIMB, COS_CLIMB, SIN_LOCK, COS_LOCK, COS_TERM,
           COS_LOSE);
    }
```

See APPENDIX I for a complete source code listing.

### 3.15.9 COS\_UNGUIDE

COS\_UNGUIDE is a constant defining the cosine of the delta pitch angle for an unguided maverick missile

#### 3.15.9.1 Initialization

The constant is initialized during execution of the CSU missile\_maverick\_init, called by CSC weapons\_init. Execution of the CSU missile\_maverick\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

```
#define COS_UNGUIDE maverick_miss_char[ 8]
```

### 3,15.9.2 Usage

During real-time execution, this constant is not recomputed.

### 3.15.9.2.1 Algorithm

COS\_UNGUIDE is used to compute the missile flyout path by a call to the CSC missile\_hellfire\_fly.

```
/*/

* Find the target point to which the missile is to fly. The missile ignores

* any targets until it is armed.

/*/

if (time < MAVERICK_ARM_TIME)

missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,

SIN_CLIMB, COS_CLIMB, SIN_LOCK,

COS_LOCK, COS_TERM, COS_LOSE);

else

{

TObjectP object = mvptr -> object_being_tracked;

/*/

* Try to find a target. If one is found, fly towards it in the

* proper trajectory, otherwise, fly in a search trajectory.

/*/

if (object != NO_OBJECT)

{

VECTOR target_location;
```

See APPENDIX I for a complete source code listing.

### 3.15.10 SIN\_CLIMB

SIN\_CLIMB is a constant defining the sine of the delta pitch angle for a climbing maverick missile

#### 3.15.10.1 Initialization

The constant is initialized during execution of the CSU missile\_maverick\_init, called by CSC weapons\_init. Execution of the CSU missile\_maverick\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

#define SIN\_CLIMB maverick\_miss\_char[ 9]

### 3.15.10.2 Usage

During real-time execution, this constant is not recomputed.

### 3.15.10.2.1 Algorithm

SIN\_CLIMB is used to compute the missile flyout path by a call to the CSC missile\_maverick\_fly.

```
* Find the target point to which the missile is to fly. The missile ignores
* any targets until it is armed.
  if (time < MAVERICK_ARM_TIME)
     missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
                        SIN_CLIMB, COS_CLIMB, SIN_LOCK,
                        COS_LOCK, COS_TERM, COS_LOSE);
 else
     TObjectP object = mvptr -> object_being_tracked;
    Try to find a target. If one is found, fly towards it in the
    proper trajectory, otherwise, fly in a search trajectory.
    if (object != NO_OBJECT)
        VECTOR target_location;
        GetLocationOfTObject (object, target_location);
       mvptr->target_vehicle_id = object -> var.vehicleID;
        missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                              COS_UNGUIDE, SIN_CLIMB, COS_CLIMB,
                              SIN_LOCK, COS_LOCK, COS_TERM,
                              COS LOSE);
    }
    else
        mvptr->target_vehicle_id.vehicle = vehicleIrrelevant;
        if (TrackAcquire (mvptr -> sensor_id, veh_list, mptr -> location,
                      mptr -> orientation[1]) < 0)
            printf ("missile_maverick_fly: TrackAcquire: %s\n",
                  TrackErrString ());
        missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
            SIN_CLIMB, COS_CLIMB, SIN_LOCK, COS_LOCK, COS_TERM,
          COS LOSE);
    }
  }
```

See APPENDIX I for a complete source code listing.

#### 3.15.11 COS\_CLIMB

COS\_CLIMB is a constant defining the cosine of the delta pitch angle for a climbing maverick missile

### 3.15.11.1 Initialization

The constant is initialized during execution of the CSU missile\_maverick\_init, called by CSC weapons\_init. Execution of the CSU missile\_maverick\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

```
#define COS_CLIMB maverick_miss_char[10]
```

### 3.15.11.2 Usage

During real-time execution, this constant is not recomputed.

## 3.15.11.2.1 Algorithm

COS\_CLIMB is used to compute the missile flyout path by a call to the CSC missile\_maverick\_fly.

```
/*/

* Find the target point to which the missile is to fly. The missile ignores

* any targets until it is armed.

/*/

if (time < MAVERICK_ARM_TIME)

missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,

SIN_CLIMB, COS_CLIMB, SIN_LOCK,

COS_LOCK, COS_TERM, COS_LOSE);

else

{

TObjectP object = mvptr -> object_being_tracked;

/*/

* Try to find a target. If one is found, fly towards it in the

* proper trajectory, otherwise, fly in a search trajectory.

/*/

if (object != NO_OBJECT)

{

VECTOR target_location;
```

```
GetLocationOfTObject (object, target_location);
   mvptr->target_vehicle_id = object -> var.vehicleID;
   missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                          COS_UNGUIDE, SIN_CLIMB, COS_CLIMB,
                          SIN_LOCK, COS_LOCK, COS_TERM,
                          COS LOSE);
}
else
    mvptr->target_vehicle_id.vehicle = vehicleIrrelevant;
    if (TrackAcquire (mvptr -> sensor_id, veh_list, mptr -> location,
                 mptr -> orientation[1]) < 0)
       printf ("missile_maverick_fly: TrackAcquire: %s\n",
              TrackErrString ());
    missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
       SIN_CLIMB, COS_CLIMB, SIN_LOCK, COS_LOCK, COS_TERM,
      COS_LOSE);
}
```

### 3.15.12 SIN\_LOCK

SIN\_LOCK is a constant defining the sine of the lock cone angle for a locked-on mayerick **mis**sile

#### 3.15.12.1 Initialization

The constant is initialized during execution of the CSU missile\_maverick\_init, called by CSC weapons\_init. Execution of the CSU missile\_maverick\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

#define SIN\_LO**CK** maverick\_miss\_char[11]

# 3.15.12.2 Usage

During real-time execution, this constant is not recomputed.

### 3.15.12.2.1 Algorithm

SIN\_LOCK is used to compute the missile flyout path by a call to the CSC missile\_maverick\_fly.

```
/*/
* Find the target point to which the missile is to fly. The missile ignores
* any targets until it is armed.
/*/
  if (time < MAVERICK_ARM_TIME)
     missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
                        SIN_CLIMB, COS_CLIMB, SIN_LOCK,
                        COS LOCK, COS_TERM, COS_LOSE);
  else
 {
     TObjectP object = mvptr -> object_being_tracked;
    Try to find a target. If one is found, fly towards it in the
    proper trajectory, otherwise, fly in a search trajectory.
    if (object != NO_OBJECT)
        VECTOR target_location;
        GetLocationOfTObject (object, target_location);
       mvptr->target_vehicle_id = object -> var.vehicleID;
        missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                               COS_UNGUIDE, SIN_CLIMB, COS_CLIMB,
                               SIN LOCK, COS LOCK, COS_TERM,
                               COS_LOSE);
    }
    else
        mvptr->target_vehicle_id.vehicle = vehicleIrrelevant;
        if (TrackAcquire (mvptr -> sensor_id, veh_list, mptr -> location,
                      mptr -> orientation[1]) < 0
            printf ("missile_maverick_fly: TrackAcquire: %s\n",
                  TrackErrString ());
        missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
            SIN_CLIMB, COS_CLIMB, SIN_LOCK, COS_LOCK, COS_TERM,
          COS LOSE);
    }
  }
```

#### 3.15.13 COS\_LOCK

COS\_LOCK is a constant defining the cosine of the lock cone angle for a locked-on maverick missile

#### 3.15.13.1 Initialization

The constant is initialized during execution of the CSU missile\_maverick\_init, called by CSC weapons\_init. Execution of the CSU missile\_maverick\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

```
#define COS_LOCK maverick_miss_char[12]
```

#### 3.15.13.2 Usage

During real-time execution, this constant is not recomputed.

### 3.15.13.2.1 Algorithm

COS\_LOCK is used to compute the missile flyout path by a call to the CSC missile maverick\_fly.

```
/*/

* Find the target point to which the missile is to fly. The missile ignores

* any targets until it is armed.

/*/

if (time < MAVERICK_ARM_TIME)

missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,

SIN_CLIMB, COS_CLIMB, SIN_LOCK,

COS_LOCK, COS_TERM, COS_LOSE);

else
{

TObjectP object = mvptr -> object_being_tracked;

/*/

* Try to find a target. If one is found, fly towards it in the

* proper trajectory, otherwise, fly in a search trajectory.

/*/

if (object != NO_OBJECT)

{

VECTOR target_location;
```

### 3.15.14 COS\_TERM

COS\_TERM is a constant defining the cosine of the terminal pitch angle for a locked-on mayerick missile

#### 3.15.14.1 Initialization

The constant is initialized during execution of the CSU missile\_maverick\_init, called by CSC weapons\_init. Execution of the CSU missile\_maverick\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

#define COS\_TERM maverick\_miss\_char[13]

### 3.15.14.2 Usage

During real-time execution, this constant is not recomputed.

#### 3.15.14.2.1 Algorithm

COS\_TERM is used to compute the missile flyout path by a call to the CSC missile\_maverick\_fly.

```
/*/
* Find the target point to which the missile is to fly. The missile ignores
* any targets until it is armed.
/*/
  if (time < MAVERICK_ARM_TIME)
     missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
                        SIN_CLIMB, COS_CLIMB, SIN_LOCK,
                        COS_LOCK, COS_TERM, COS_LOSE);
 else
 {
      TObjectP object = mvptr -> object_being_tracked;
    Try to find a target. If one is found, fly towards it in the
    proper trajectory, otherwise, fly in a search trajectory.
/*/
    if (object != NO_OBJECT)
        VECTOR target_location;
        GetLocationOfTObject (object, target_location);
       mvptr->target_vehicle_id = object -> var.vehicleID;
        missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                              COS_UNGUIDE, SIN_CLIMB, COS_CLIMB,
                               SIN_LOCK, COS_LOCK, COS_TERM,
                               COS_LOSE);
    }
    else
        mvptr->target_vehicle_id.vehicle = vehicleIrrelevant;
        if (TrackAcquire (mvptr -> sensor_id, veh_list, mptr -> location,
                      mptr \rightarrow orientation[1] < 0
            printf ("missile_maverick_fly: TrackAcquire: %s\n",
                  TrackErrString ());
        missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
           SIN_CLIMB, COS_CLIMB, SIN_LOCK, COS_LOCK, COS_TERM,
          COS LOSE);
    }
  }
```

#### 3.15.15 COS\_LOSE

COS\_LOSE is a constant defining the cosine of the pitch angle for a loss-of-lock-on maverick missile

#### 3.15.15.1 Initialization

The constant is initialized during execution of the CSU missile\_maverick\_init, called by CSC weapons\_init. Execution of the CSU missile\_maverick\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.15. - Maverick Missile Characteristics Data Array for a summary of the constants data.

```
#define COS_LOSE maverick_miss_char[14]
```

### 3.15.15.2 Usage

During real-time execution, this constant is not recomputed.

### 3.15.15.2.1 Algorithm

COS\_LOSE is used to compute the missile flyout path by a call to the CSC missile\_maverick\_fly.

```
/*/

* Find the target point to which the missile is to fly. The missile ignores

* any targets until it is armed.

/*/

if (time < MAVERICK_ARM_TIME)

missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,

SIN_CLIMB, COS_CLIMB, SIN_LOCK,

COS_LOCK, COS_TERM, COS_LOSE);

else

{

TObjectP object = mvptr -> object_being_tracked;

/*/

* Try to find a target. If one is found, fly towards it in the

* proper trajectory, otherwise, fly in a search trajectory.

/*/

if (object != NO_OBJECT)

{

VECTOR target_location;
```

```
GetLocationOfTObject (object, target location);
   mvptr->target_vehicle_id = object -> var.vehicleID;
    missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                          COS_UNGUIDE, SIN_CLIMB, COS_CLIMB,
                          SIN LOCK, COS LOCK, COS_TERM,
                          COS LOSE);
}
else
    mvptr->target_vehicle_id.vehicle = vehicleIrrelevant;
    if (TrackAcquire (mvptr -> sensor_id, veh_list, mptr -> location,
                 mptr -> orientation[1]) < 0
        printf ("missile_maverick_fly: TrackAcquire: %s\n",
              TrackErrString ());
   missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
       SIN_CLIMB, COS_CLIMB, SIN_LOCK, COS_LOCK, COS_TERM,
      COS_LOSE);
```

### 3.16 Maverick\_miss\_poly\_deg

The maverick\_miss\_poly\_deg array consists of values of the degree of each polynomial equation used to compute the burn speed and the coast speed for the Maverick missile.

# 3.16.1 MAVERICK\_BURN\_SPEED\_DEG

MAVERICK\_BURN\_SPEED\_DEG is a constant defining the polynomial degree for the Maverick missile burn speed coefficient data array. MAVERICK\_BURN\_SPEED\_DEG is the first element of the maverick\_miss\_poly\_deg array.

#### 3.16.1.1 Initialization

The constant is initialized during execution of the CSU missile\_maverick\_init, called by CSC weapons\_init. Execution of the CSU missile\_maverick\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.16. - Maverick Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define MAVERICK_BURN_SPEED_DEG maverick_miss_poly_deg[ 0]
```

### 3.16.1.2 Usage

During real-time execution, this variable is not recomputed. The maximum value for MAVERICK\_BURN\_SPEED\_DEG is 4, especially, the declared size of the maverick burn speed\_coeff array is 5.

### 3.16.1.2.1 Algorithm

MAVERICK\_BURN\_SPEED\_DEG is used to compute the maverick missile speed at launch using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_maverick\_fire. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

MAVERICK\_BURN\_SPEED\_DEG is used to compute the maverick missile speed during powered flight [burn] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_maverick\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

### 3.16.2 MAVERICK\_COAST\_SPEED\_DEG

MAVERICK\_COAST\_SPEED\_DEG is a constant defining the polynomial degree for the maverick missile coast speed coefficient data array. MAVERICK\_COAST\_SPEED\_DEG is the second element of the maverick\_miss\_poly\_deg array.

#### 3.16.2.1 Initialization

The constant is initialized during execution of the CSU missile\_maverick\_init, called by CSC weapons\_init. Execution of the CSU missile\_maverick\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.16. - Maverick Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define MAVERICK_COAST_SPEED_DEG maverick_miss_poly_deg[ 1]
```

### 3.16.2.2 Usage

During real-time execution, this variable is not recomputed. The maximum value for MAVERICK\_COAST\_SPEED\_DEG is 4, especially, the declared size of the maverick\_coast\_speed\_coeff array is 5.

# 3.16.2.2.1 Algorithm

MAVERICK\_COAST\_SPEED\_DEG is used to compute the maverick missile speed during unpowered flight [coast] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_maverick\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

### 3.17 Maverick\_burn\_speed\_coeff

The maverick\_burn\_speed\_coeff array consists of the coefficients for a polynomial equation defining the Maverick missile burn speed with respect to time in the form using the Newton-Raphson method.

#### 3.17.1 Initialization

The maverick\_burn\_speed\_coeff array is initialized during execution of the CSU missile\_maverick\_init, called by CSC weapons\_init. Execution of the CSU missile\_maverick\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.17. - Maverick Missile Burn Speed Data Array for a summary of the array data.

The array has a maximum size of 5 elements.

### 3.17.2 Usage

During real-time execution, this array is not recomputed. MAVERICK\_BURN\_SPEED\_DEG determines the number of elements of the array to be used in the polynomial evaluation.

### 3.17.2.1 Algorithm

The maverick\_burn\_speed\_coeff array is used to compute the maverick missile speed at launch using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_maverick\_fire. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

The maverick\_burn\_speed\_coeff array is used to compute the maverick missile speed during powered flight [burn] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_maverick\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

#### 3.18 Maverick\_coast\_speed\_coeff

The maverick\_coast\_speed\_coeff array consists of the coefficients for a polynomial equation defining the Maverick missile coast speed with respect to time in the form using the Newton-Raphson method.

#### 3.18.1 Initialization

The maverick\_coast\_speed\_coeff array is initialized during execution of the CSU missile\_maverick\_init, called by CSC weapons\_init. Execution of the CSU missile\_maverick\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.18. - Maverick Missile Coast Speed Data Array for a summary of the constants data.

The array has a maximum size of 5 elements.

#### 3.18.2 Usage

During real-time execution, this array is not recomputed. MAVERICK\_COAST\_SPEED\_DEG determines the number of elements of the array to be used in the polynomial evaluation.

#### 3.18.2.1 Algorithm

The maverick\_coast\_speed\_coeff array is used to compute the maverick missile speed during unpowered flight [coast] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_maverick\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

### 3.19 Stinger\_miss\_char

The stinger\_miss\_char array consists of characteristics and parameters describing a Stinger missile system and its performance constraints.

### 3.19.1 STINGER\_BURNOUT\_TIME

STINGER\_BURNOUT\_TIME is a constant defining the time of powered flight for stinger missile in ticks.

#### 3.19.1.1 Initialization

The constant is initialized during execution of the CSU missile\_stinger\_init, called by CSC weapons\_init. Execution of the CSU missile\_stinger\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.19. - Stinger Missile Characteristics Data Array for a summary of the constants data.

```
#define STINGER_BURNOUT_TIME stinger_miss_char[ 0]
```

# 3.19.1.2 Usage

During real-time execution, this constant is not recomputed.

# 3.19.1.2.1 Algorithm

STINGER\_BURNOUT\_TIME is used to control computation of the missile flyout speed and the cosine of the maximum allowed turn by a call to the CSC missile\_stinger\_fly.

```
/*/
* Find the current missile speed and the cosine of the maximum
      allowed turn angle. The equations used are different before and
      after motor burnout.
/*/
  if (time < STINGER_BURNOUT_TIME)</pre>
     mptr->speed = missile_util_eval_poly (STINGER_BURN_SPEED_DEG,
          stinger_burn_speed_coeff, time) + mptr->init_speed;
 else
     mptr->speed = missile_util_eval_poly (STINGER_COAST_SPEED_DEG,
          stinger_coast_speed_coeff, time) + mptr->init_speed;
 }
/*/
* Note that this is a temporary method of finding turn angle.
 ' mptr->cos_max_turn[0] = cos (sqrt (mptr->speed / (SPEED_0 +
       mptr->init_speed)) * THETA_0);
/*/
 Try to find a target. If one is found, fly towards it in the
* proper trajectory, otherwise, fly in a straight line.
/*/
   target = near get preferred veh near vector (&(sptr->target_vehicle_id),
       veh_list, mptr->location, mptr->orientation[1],
       STINGER_LOCK_THRESHOLD);
  if( max_range_limit > 0 &&
     kinematics_range_squared (veh_kinematics, mptr->location) >
    max_range_squared)
     missile_target_ground( mptr );
  else if (target != NULL)
     sptr->target_vehicle_id = target->vehicleID;
     if (time < STINGER_BURNOUT_TIME)
        missile_target_intercept_pre_burnout (mptr, target,
             sptr->stinger_burn_range_coeff, STINGER_BURNOUT_TIME,
           STINGER_BURN_SPEED_DEG + 1,
           sptr->stinger_coast_range_coeff,
           sptr->stinger_coast_range_2_coeff,
           STINGER_COAST_SPEED_DEG + 1);
```

### 3.19.2 STINGER\_MAX\_FLIGHT\_TIME

STINGER\_MAX\_FLIGHT\_TIME is a constant defining the maximum flight time for the stinger missile assumed in ticks.

#### 3.19.2.1 Initialization

The constant is initialized during execution of the CSU missile\_stinger\_init, called by CSC weapons\_init. Execution of the CSU missile\_stinger\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.19. - Stinger Missile Characteristics Data Array for a summary of the constants data.

```
#define STINGER_MAX_FLIGHT_TIME stinger_miss_char[ 1]
```

#### 3.19.2.2 Usage

During real-time execution, this constant is not recomputed.

# 3.19.2.2.1 Algorithm

STINGER\_MAX\_FLIGHT\_TIME is used to initialize the maximum flight time for an individual stinger missile by a call to the CSU missile\_stinger\_init.

### 3.19.3 STINGER\_LOCK\_THRESHOLD

STINGER\_LOCK\_THRESHOLD is a constant defining the cosine squared of the lock threshold angle for the stinger missile.

#### 3.19.3.1 Initialization

The constant is initialized during execution of the CSU missile\_stinger\_init, called by CSC weapons\_init. Execution of the CSU missile\_stinger\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.19. - Stinger Missile Characteristics Data Array for a summary of the constants data.

```
#define STINGER_LOCK_THRESHOLD stinger_miss_char[ 2]
```

#### 3.19.3.2 Usage

During real-time execution, this constant is not recomputed.

### 3.19.3.2.1 Algorithm

STINGER\_LOCK\_THRESHOLD is used to compute a target during prelaunch by a call to the CSU missile\_stinger\_pre\_launch.

```
/*/

* Try to find a target.

/*/

target = near_get_preferred_veh_near_vector (&(sptr->target_vehicle_id),

veh_list, launch_point, launch_to_world[1],

STINGER_LOCK_THRESHOLD);
```

STINGER\_LOCK\_THRESHOLD is used to compute the target by a call to the CSC missile\_stinger\_fly.

```
/*/

* Try to find a target. If one is found, fly towards it in the

* proper trajectory, otherwise, fly in a straight line.

/*/

target = near_get_preferred_veh_near_vector (&(sptr->target_vehicle_id),

veh_list, mptr->location, mptr->orientation[1],

STINGER_LOCK_THRESHOLD);
```

See APPENDIX K for a complete source code listing.

#### 3.19.4 SPEED\_0

SPEED\_0 is a constant defining the reference turn speed used to compute the ratio for the maximum turn angle.

#### 3.19.4.1 Initialization

The constant is initialized during execution of the CSU missile\_stinger\_init, called by CSC weapons\_init. Execution of the CSU missile\_stinger\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.19. - Stinger Missile Characteristics Data Array for a summary of the constants data.

		1
#define SPEED_0	stinger_miss_char[ 3]	

### 3.19.4.2 Usage

During real-time execution, this constant is not recomputed.

# 3.19.4.2.1 Algorithm

SPEED\_0 is used to compute the maximum turn angle for an individual stinger missile by a call to the CSC missile\_stinger\_fly.

```
/*/
* Note that this is a temporary method of finding turn angle.
  mptr->cos_max_turn[0] = cos (sqrt (mptr->speed / (SPEED_0 +
       mptr->init_speed)) * THETA_0);
```

#### 3.19.5 THETA\_0

default is 15.0 deg/sec

THETA\_0 is a constant defining the reference maximum turn angle which is scaled for speed.

#### Initialization 3.19.5.1

The constant is initialized during execution of the CSU missile\_stinger\_init, called by CSC weapons\_init. Execution of the CSU missile\_stinger\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.19. - Stinger Missile Characteristics Data Array for a summary of the constants data.

```
stinger miss_char[4]
#define THETA_0
```

#### Usage 3.19.5.2

During real-time execution, this constant is not recomputed.

#### Algorithm 3.19.5.2.1

THETA\_0 is used to compute the maximum turn angle for an individual stinger missile by a call to the CSC missile\_stinger\_fly.

```
* Note that this is a temporary method of finding turn angle.
/*/
 mptr->cos_max_turn[0] = cos (sqrt (mptr->speed / (SPEED_0 +
       mptr->init_speed)) * THETA_0);
```

See APPENDIX K for a complete source code listing.

### 3.19.6 INVEST\_DIST\_SQ

The INVEST\_DIST\_SQ is a constant defining the area at a maximum speed of less than 100 m/sec.

#### 3.19.6.1 Initialization

The INVEST\_DIST\_SQ is initialized during execution of the CSU missile\_stinger\_init, called by CSU weapons\_init. See TABLE 5.1.19. - Stinger Missile Characteristics Data Array for a summary of the constant.

```
#define INVEST_DIST_SQ stinger_miss_char[5]
```

# 3.19.6.2 Usage

During real-time execution, this constant is not recomputed.

# 3,19.6.2.1 Algorithm

INVEST\_DIST\_SQ is used to compute detonation of the proximity fuze by a call in the CSU missile\_stinger\_fly.

See APPENDIX K for a complete source code listing.

# 3.19.7 FUZE\_DIST\_SQ

FUZE\_DIST\_SQ is a constant defining the square of the radius of the cylinder describing the flechettes fly in a cylinder with a radius of 20 meters and a length of 750 meters

#### 3.19.7.1 Initialization

The FUZE\_DIST\_SQ is initialized during execution of the CSU missile\_stinger\_init, called by CSU weapons\_init. See TABLE 5.1.19. - Stinger Missile Characteristics Data Array for a summary of the constant.

```
#define FUZE_DIST_SQ stinger_miss_char[ 6]
```

### 3.19.7.2 Usage

During real-time execution, this constant is not recomputed.

### 3.19.7.2.1 Algorithm

FUZE\_DIST\_SQ is used to compute detonation of the proximity fuze by a call in the CSU missile\_stinger\_fly.

See APPENDIX K for a complete source code listing.

# 3.20 Stinger\_miss\_poly\_deg

The stinger\_miss\_poly\_deg array consists of values of the degree of each polynomial equation used to compute the burn speed and the coast speed for the Stinger missile.

# 3.20.1 STINGER\_BURN\_SPEED\_DEG

STINGER\_BURN\_SPEED\_DEG is a constant defining the polynomial degree for the Stinger missile burn speed coefficient data array.

STINGER\_BURN\_SPEED\_DEG is the first element of the stinger\_miss\_poly\_deg array. STINGER\_BURN\_SPEED\_DEG is also known as stinger\_miss\_poly\_deg[ 0].

### 3.20.1.1 Initialization

The constant is initialized during execution of the CSU missile\_stinger\_init, called by CSC weapons\_init. Execution of the CSU missile\_stinger\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.20. - Stinger Missile Polynomial Degree Data Array for a summary of the constants data.

#### 3.20.1.2 Usage

During real-time execution, this variable is not recomputed. The value for stinger\_miss\_poly\_deg[0] is 1, especially, the declared size of the stinger\_burn\_speed\_coeff array is 2. This value cannot be changed with a change to the source code because of other dependencies in the code structure.

# 3,20.1.2.1 Algorithm

STINGER\_BURN\_SPEED\_DEG is used to compute the stinger missile speed at launch using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_stinger\_fire. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

STINGER\_BURN\_SPEED\_DEG is used to compute the stinger missile speed during powered flight [burn] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_stinger\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
* Set _mptr_ and _time_. These values are created mostly for increased
* readablity.
 mptr = &(sptr->mptr);
  time = mptr->time;
* Find the current missile speed and the cosine of the maximum allowed
* turn angle. The equations used are different before and after
* motor burnout.
/*/
  if (time < STINGER_BURNOUT_TIME)
      mptr->speed = missile_util_eval_poly (STINGER_BURN_SPEED_DEG,
          stinger_burn_speed_coeff, time) + mptr->init_speed;
  else
      mptr->speed = missile_util_eval_poly (STINGER_COAST_SPEED_DEG,
          stinger_coast_speed_coeff, time) + mptr->init_speed;
  }
```

See APPENDIX K for a complete source code listing.

# 3.20.2 STINGER\_COAST\_SPEED\_DEG

STINGER\_COAST\_SPEED\_DEG is a constant defining the polynomial degree for the stinger missile coast speed coefficient data array. STINGER\_COAST\_SPEED\_DEG is the second element of the stinger\_miss\_poly\_deg array. STINGER\_COAST\_SPEED\_DEG is also known as stinger\_miss\_poly\_deg[ 1].

### 3.20.2.1 Initialization

The constant is initialized during execution of the CSU missile\_stinger\_init, called by CSC weapons\_init. Execution of the CSU missile\_stinger\_init is normally done only once during CSCI initialization and is performed

sequentially. See TABLE 5.1.20. - Stinger Missile Polynomial Degree Data Array for a summary of the constants data.

#### 3.20.2.2 Usage

During real-time execution, this variable is not recomputed. The value for STINGER\_COAST\_SPEED\_DEG is 3, especially, the declared size of the stinger\_coast\_speed\_coeff array is 4. This value cannot be changed with a change to the source code because of other dependencies in the code structure.

# 3.20.2.2.1 Algorithm

STINGER\_COAST\_SPEED\_DEG is used to compute the stinger missile speed during unpowered flight [coast] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_stinger\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/;*/
* Set _mptr_ and _time_. These values are created mostly for increased
readablity.
/*/
  mptr = &(sptr->mptr);
  time = mptr->time;
/*/
* Find the current missile speed and the cosine of the maximum allowed
* turn angle. The equations used are different before and after
* motor burnout.
/*/
   if (time < STINGER_BURNOUT_TIME)
      mptr->speed = missile_util_eval_poly (STINGER_BURN_SPEED_DEG,
          stinger_burn_speed_coeff, time) + mptr->init_speed;
  else
      mptr->speed = missile_util_eval_poly (STINGER_COAST_SPEED_DEG,
          stinger_coast_speed_coeff, time) + mptr->init_speed;
  }
```

See APPENDIX K for a complete source code listing.

# 3.21 Stinger\_burn\_speed\_coeff

The stinger\_burn\_speed\_coeff array consists of the coefficients for a polynomial equation defining the Stinger missile burn speed with respect to time in the form using the Newton-Raphson method.

#### 3.21.1 Initialization

The stinger\_burn\_speed\_coeff array is initialized during execution of the CSU missile\_stinger\_init, called by CSC weapons\_init. Execution of the CSU missile\_stinger\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.21. - Stinger Missile Burn Speed Data Array for a summary of the array data.

The array has a maximum size of 5 elements.

### 3.21.2 Usage

During real-time execution, this array is not recomputed. STINGER\_BURN\_SPEED\_DEG determines the number of elements of the array to be used in the polynomial evaluation.

# 3.21.2.1 Algorithm

The stinger\_burn\_speed\_coeff array is used to compute the stinger missile speed at launch using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_stinger\_fire. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/

* Get a pointer to the generic elements of the STINGER missile. This

* improves code readability.

/*/

mptr = &(sptr->mptr);

/*/

* Set the initial time, location, orientation and speed of the generic

* missile.

/*/

mptr->time = 0.0;
vec_copy (launch_point, mptr->location);
mat_copy (launch_to_world, mptr->orientation);
```

The stinger\_burn\_speed\_coeff array is used to compute the stinger missile speed during powered flight [burn] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_stinger\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/

* Set _mptr_ and _time_. These values are created mostly for increased

* readablity.
/*/

, mptr = &(sptr->mptr);
  time = mptr->time;
/*/

* Find the current missile speed and the cosine of the maximum allowed

* turn angle. The equations used are different before and after

* motor burnout.
/*/

if (time < STINGER_BURNOUT_TIME)
{
  mptr->speed = missile_util_eval_poly (STINGER_BURN_SPEED_DEG, stinger_burn_speed_coeff, time) + mptr->init_speed;
}
else
{
  mptr->speed = missile_util_eval_poly (STINGER_COAST_SPEED_DEG, stinger_coast_speed_coeff, time) + mptr->init_speed;
}
```

See APPENDIX K for a complete source code listing.

# 3.22 Stinger\_coast\_speed\_coeff

The stinger\_coast\_speed\_coeff array consists of the coefficients for a polynomial equation defining the Stinger missile coast speed with respect to time in the form using the Newton-Raphson method.

#### 3.22.1 Initialization

The stinger\_coast\_speed\_coeff array is initialized during execution of the CSU missile\_stinger\_init, called by CSC weapons\_init. Execution of the CSU missile\_stinger\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.22. - Stinger Missile Coast Speed Data Array for a summary of the constants data.

The array has a maximum size of 5 elements.

#### 3.22.2 Usage

During real-time execution, this array is not recomputed. STINGER\_COAST\_SPEED\_DEG determines the number of elements of the array to be used in the polynomial evaluation.

### 3.22.2.1 Algorithm

The stinger\_coast\_speed\_coeff array is used to compute the stinger missile speed during unpowered flight [coast] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_stinger\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/

* Set _mptr_ and _time_. These values are created mostly for increased

* readablity.

/*/

mptr = &(sptr->mptr);

time = mptr->time;

/*/

* Find the current missile speed and the cosine of the maximum allowed

* turn angle. The equations used are different before and after

* motor burnout.

/*/

if (time < STINGER_BURNOUT_TIME)

{

mptr->speed = missile_util_eval_poly (STINGER_BURN_SPEED_DEG, stinger_burn_speed_coeff, time) + mptr->init_speed;
}
```

#### 3.23 Tow\_miss\_char

The tow\_miss\_char array consists of characteristics and parameters describing a TOW missile system and its performance constraints.

#### 3.23.1 TOW\_BURNOUT\_TIME

TOW\_BURNOUT\_TIME is a constant defining the time of powered flight for tow missile in ticks.

### 3.23.1.1 Initialization

The constant is initialized during execution of the CSU missile\_tow\_init, called by CSC weapons\_init. Execution of the CSU missile\_tow\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.23. - Tow Missile Characteristics Data Array for a summary of the constants data.

```
#define TOW_BURNOUT_TIME tow_miss_char[ 0]
```

#### 3.23.1.2 Usage

During real-time execution, this constant is not recomputed.

# 3.23.1.2.1 Algorithm

TOW\_BURNOUT\_TIME is used to control computation of the missile flyout speed and the cosines of the maximum allowed turn angles in each direction by a call to the CSC missile\_tow\_fly.

```
/*/

* Find the current missile speed and the cosines of the maximum

* allowed turn angles in each direction. The equations used are

* different before and after motor burnout.

/*/
```

# 3.23.2 TOW\_RANGE\_LIMIT\_TIME

TOW\_RANGE\_LIMIT\_TIME is a constant defining the range limit time for the tow missile in ticks; at this point the wire is cut, but the missile is allowed to fly to the maximum flight time.

### 3.23.2.1 Initialization

The constant is initialized during execution of the CSU missile\_tow\_init, called by CSC weapons\_init. Execution of the CSU missile\_tow\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.23. - Tow Missile Characteristics Data Array for a summary of the constants data.

#define TOW\_RANGE\_LIMIT\_TIME tow\_miss\_char[ 1]

### 3.23.2.2 Usage

During real-time execution, this constant is not recomputed.

# 3.23.2.2.1 Algorithm

TOW\_RANGE\_LIMIT\_TIME is used to control the wire cut for an individual tow missile by a call to the CSU missile\_tow\_fly

```
/*/

* If the missile has reached its maximum range (not the maximum distance

* its allowed to fly), cut the wire.

/*/

#ifdef notdeff

if ((time > TOW_RANGE_LIMIT_TIME) && !tptr->wire_is_cut)

tptr->wire_is_cut = TRUE;

#endif

, if (!tptr->wire_is_cut &&

((time > TOW_RANGE_LIMIT_TIME) | |

(max_range_limit > 0 &&

kinematics_range_squared (veh_kinematics, mptr->location) >

max_range_squared)))

tptr->wire_is_cut = TRUE;
```

See APPENDIX L for a complete source code listing.

# 3.23.3 TOW\_MAX\_FLIGHT\_TIME

TOW\_MAX\_FLIGHT\_TIME is a constant defining the maximum flight time for the tow missile assumed in ticks; cosine of the max turn is greater than 1.0 beyond this point.

# 3.23.3.1 Initialization

The constant is initialized during execution of the CSU missile\_tow\_init, called by CSC weapons\_init. Execution of the CSU missile\_tow\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.23. - Tow Missile Characteristics Data Array for a summary of the constants data.

```
#define TOW_MAX_FLIGHT_TIME tow_miss_char[ 2]
```

#### 3.23.3.2 Usage

During real-time execution, this constant is not recomputed.

### 3.23.3.2.1 Algorithm

TOW\_MAX\_FLIGHT\_TIME is used to initialize the maximum flight time for an individual tow missile by a call to the CSU missile\_tow\_init.

tptr->mptr.max\_flight\_time = TOW\_MAX\_FLIGHT\_TIME;

See APPENDIX L for a complete source code listing.

### 3.24 Tow\_miss\_poly\_deg

The tow\_miss\_poly\_deg array consists of values of the degree of each polynomial equation used to compute the burn speed, the coast speed, maximum cosines of turns while powered, and maximum cosines of turns while unpowered for the TOW missile.

### 3.24.1 TOW\_BURN\_SPEED\_DEG

TOW\_BURN\_SPEED\_DEG is a constant defining the polynomial degree for the tow missile burn speed coefficient data array.

#### 3.24.1.1 Initialization

The constant is initialized during execution of the CSU missile\_tow\_init, called by the CSC weapons\_init. Execution of the CSU missile\_tow\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.24. - Tow Missile Polynomial Degree Data Array for a summary of the constants data.

#define TOW\_BURN\_SPEED\_DEG tow\_miss\_poly\_deg[0]

### 3.24.1.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for TOW\_BURN\_SPEED\_DEG is 4, especially, the declared size of the tow burn\_speed\_coeff is 5.

# 3.24.1.2.1 Algorithm

TOW\_BURN\_SPEED\_DEG is used to compute the tow missile speed at launch using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_tow\_fire. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/

* Set the initial time, location, orientation, and speed of the generic

* missile.

/*/

mptr->time = 0.0;

vec_copy (launch_point, mptr->location);

mat_copy (loc_sight_to_world, mptr->orientation);

mptr->speed = launch_speed +

(speed_factor * missile_util_eval_poly (TOW_BURN_SPEED_DEG,

tow_burn_speed_coeff, 0.0));

mptr->init_speed = launch_speed;
```

TOW\_BURN\_SPEED\_DEG is used to compute the tow missile speed during powered flight [burn] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_tow\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

# 3.24.2 TOW\_COAST\_SPEED\_DEG

TOW\_COAST\_SPEED\_DEG is a constant defining the polynomial degree for tow missile coast speed coefficient data array

### 3.24.2.1 Initialization

The constant is initialized during execution of the CSU missile\_tow\_init, called by the CSC weapons\_init. Execution of the CSU missile\_tow\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.24. - Tow Missile Polynomial Degree Data Array for a summary of the array data.

```
#define TOW_COAST_SPEED_DEG tow_miss_poly_deg[1]
```

### 3.24.2.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for TOW\_COAST\_SPEED\_DEG is 4, especially, the declared size of the tow\_burn\_speed\_coeff is 5.

# 3.24.2.2.1 Algorithm

TOW\_COAST\_SPEED\_DEG is used to compute the tow missile speed during unpowered flight [coast] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_tow\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/

* Find the current missile speed and the cosines of the maximum

* allowed turn angles in each direction. The equations used are

* different before and after motor burnout.

/*/
```

# 3.24.3 TOW\_BURN\_TURN\_DEG

TOW\_BURN\_TURN\_DEG is a constant defining the polynomial degree for each tow missile burn turn coefficient data sub-array of the tow missile burn turn coefficient data array structure

#### 3.24.3.1 Initialization

The constant is initialized during execution of the CSU missile\_tow\_init, called by CSC weapons\_init. Execution of the CSU missile\_tow\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.24. - Tow Missile Polynomial Degree Data Array for a summary of the constant data.

```
#define TOW_BURN_TURN_DEG tow_miss_poly_deg[2]
```

# 3.24.3.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for TOW\_BURN\_TURN\_DEG is 1, especially, the declared size of the tow\_burn\_turn\_coeff is 2. Changing this constant requires a recompile because of the hard coded multi-dimension characteristic.

```
* Coefficients for the cosine of max turn polynomials before motor burnout.
* The structure MAX_COS_COEFF_ is used to store the values for the turn
* sideways, up, and down polynomials along with their order.
/*/
static MAX COS_COEFF tow_burn_turn_coeff =
              /* Order of the polynomials. */
  1,
              /* Sidewards turn. */
    0.999976868652, /* a_0 - cos(rad)/tick */
    -3.5933955e-7 /* a 1 - cos(rad)/tick**2 */
  },
             /* Upwards turn. */
    0.999960667258, /* a_0 - cos(rad)/tick */
    -3.1492328e-6 /* a 1 - cos(rad)/tick**2 */
              /* Downwards turn. */
    0.999978909989, /* a 0 - cos(rad)/tick */
    -7.8194991e-9 /* a 1 - cos(rad)/tick**2 */
};
```

### 3.24.3.2.1 Algorithm

TOW\_BURN\_TURN\_DEG is hard coded by type definition of MAX\_COS\_COEFF and is used to compute the cosine of the maximum allowed turn angle in each axis for the tow missile during powered flight [burn] using the CSU missile\_util\_cos\_coeff, and called by the CSU missile\_tow\_fly. The CSU missile\_util\_cos\_coeff uses the Newton-Raphson method to evaluate the polynomial with inputs of missile pointer, coefficient array, and time.

```
/*/

* Find the current missile speed and the cosines of the maximum

* allowed turn angles in each direction. The equations used are

* different before and after motor burnout.

/*/
```

### 3.24.4 TOW\_COAST\_TURN\_DEG

TOW\_COAST\_TURN\_DEG is a constant defining the polynomial degree for each tow missile coast turn coefficient data sub-array of the tow missile coast turn coefficient data array structure

#### 3.24.4.1 Initialization

The constant is initialized during execution of the CSU missile\_tow\_init, called by CSC weapons\_init. Execution of the CSU missile\_tow\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.24. - Tow Missile Polynomial Degree Data Array for a summary of the array data.

```
#define TOW_COAST_TURN_DEG tow_miss_poly_deg[3]
```

### 3.24.4.2 Usage

During real-time execution, this constant is not recomputed. The maximum value of each axis for TOW\_COAST\_TURN\_DEG is 3, especially, the declared size of the tow\_coast\_turn\_coeff is 4. Changing this constant requires a recompile because of the hard coded multi-dimension characteristic.

```
/*/
* Coefficients for the cosine of max turn polynomials after motor burnout.
/*/
static MAX_COS_COEFF tow_coast_turn_coeff =
              /* Order of the polynomials. */
  3,
             /* Sidewards turn. */
    0.99995112518, /* a_0 - cos(rad)/tick */
                /* a_1 - cos(rad)/tick**2 */
    8.96333e-7,
   -5.995375e-9, /* a_2 - cos(rad)/tick**3 */
    1.162225e-11 /* a_3 - cos(rad)/tick**4 */
  },
  {
             /* Upwards turn. */
    1.657779e-6, /* a_1 - cos(rad)/tick**2 */
    -8.231861e-9, /* a_2 - cos(rad)/tick**3 */
    1.381832e-11 /* a_3 - cos(rad)/tick**4 */
  },
              /* Downwards turn. */
    0.9999714014, /* a_0 - cos(rad)/tick */
    3.382077e-7, /* a_1 - cos(rad)/tick**2 */
    -1.601259e-9, /* a_2 - cos(rad)/tick**3 */
     2.623014e-12 /* a_3 - cos(rad)/tick**4 */
|};
```

# 3.24.4.2.1 Algorithm

TOW\_COAST\_TURN\_DEG is hard coded by type definition of MAX\_COS\_COEFF and is used to compute the cosine of the maximum allowed turn angle in each axis for the tow missile during unpowered flight [coast] using the CSU missile\_util\_cos\_coeff, and called by the CSU missile\_tow\_fly. The CSU missile\_util\_cos\_coeff uses the Newton-Raphson method to evaluate the polynomial with inputs of missile pointer, coefficient array, and time.

```
Find the current missile speed and the cosines of the maximum
     allowed turn angles in each direction. The equations used are
     different before and after motor burnout.
/*/
  if (time < TOW_BURNOUT_TIME)
    mptr->speed = mptr->init_speed +
      (speed_factor *
         missile_util_eval_poly (TOW_BURN_SPEED_DEG,
                    tow_burn_speed_coeff, time));
      missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
 else
    mptr->speed = mptr->init_speed +
      (speed_factor *
         missile_util_eval_poly (TOW_COAST_SPEED_DEG,
                    tow_coast_speed_coeff, time));
      missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
  }
```

# 3.25 Tow\_burn\_speed\_coeff

The tow\_burn\_speed\_coeff array consists of the coefficients for a polynomial equation defining the TOW missile burn speed with respect to time in the form using the Newton-Raphson method.

### 3.25.1 Initialization

The tow\_burn\_speed\_coeff array is initialized during execution of the CSU missile\_tow\_init, called by CSC weapons\_init. Execution of the CSU missile\_tow\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.25. - TOW Missile Burn Speed Coefficient Data Array for a summary of the array data.

The array has a maximum size of 5 elements.

# 3.25.2 Usage

During real-time execution, this array is not recomputed. TOW\_BURN\_SPEED\_DEG determines the number of elements of the array to be used in the polynomial evaluation.

### 3.25.2.1 Algorithm

Tow\_burn\_speed\_coeff is used to compute the tow missile speed at launch using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_tow\_fire. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/

* Set the initial time, location, orientation, and speed of the generic

* missile.

/*/

mptr->time = 0.0;

vec_copy (launch_point, mptr->location);

mat_copy (loc_sight_to_world, mptr->orientation);

mptr->speed = launch_speed +

(speed_factor * missile_util_eval_poly (TOW_BURN_SPEED_DEG,
tow_burn_speed_coeff, 0.0));

' mptr->init_speed = launch_speed;
```

Tow\_burn\_speed\_coeff is used to compute the tow missile speed during powered flight [burn] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_tow\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
missile_util_eval_poly (TOW_COAST_SPEED_DEG,
tow_coast_speed_coeff, time));
missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
}
```

# 3.26 Tow\_coast\_speed\_coeff

This data array consists of the coefficients for a polynomial equation defining the TOW missile coast speed with respect to time in the form using the Newton-Raphson method.

### 3.26.1 Initialization

The tow\_coast\_speed\_coeff array is initialized during execution of the CSU missile\_tow\_init, called by CSC weapons\_init. Execution of the CSU missile\_tow\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.26. - TOW Missile Coast Speed Coefficient Data Array for a summary of the array data.

The array has a maximum size of 5 elements.

## 3.26.2 Usage

During real-time execution, this array is not recomputed. TOW\_COAST\_SPEED\_DEG determines the number of elements of the array to be used in the polynomial evaluation.

# 3.26.2.1 Algorithm

Tow\_coast\_turn\_coeff is used to compute the tow missile speed during unpowered flight [coast] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_tow\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/

* Find the current missile speed and the cosines of the maximum

* allowed turn angles in each direction. The equations used are

* different before and after motor burnout.

/*/

if (time < TOW_BURNOUT_TIME)

{
```

### 3.27 Tow\_burn\_turn\_coeff

The tow\_burn\_turn\_coeff two-dimensional data array consists of the coefficients for three polynomial equations [sidewards, upwards, and downwards movement] defining the TOW missile maximum cosine of turn while powered with respect to time in the form using the Newton-Raphson method.

#### 3.27.1 Initialization

The tow\_burn\_turn\_coeff array is initialized during execution of the CSU missile\_tow\_init, called by CSC weapons\_init. Execution of the CSU missile\_tow\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.27. - TOW Missile Burn Turn Coefficients Data Array for a summary of the array data.

The array has a maximum size of 3 by 2 elements.

```
/*/

* Coefficients for the cosine of max turn polynomials before motor burnout.

* The structure _MAX_COS_COEFF_ is used to store the values for the turn

* sideways, up, and down polynomials along with their order.

/*/

static MAX_COS_COEFF tow_burn_turn_coeff =

{
    /* Order of the polynomials. */
```

```
{
    /* Sidewards turn. */
    0.999976868652, /* a_0 - cos(rad)/tick */
    -3.5933955e-7 /* a_1 - cos(rad)/tick**2 */
},
{
    /* Upwards turn. */
    0.999960667258, /* a_0 - cos(rad)/tick */
    -3.1492328e-6 /* a_1 - cos(rad)/tick**2 */
},
{
    /* Downwards turn. */
    0.999978909989, /* a_0 - cos(rad)/tick */
    -7.8194991e-9 /* a_1 - cos(rad)/tick**2 */
};
```

Changing this constant requires a recompile because of the hard coded multidimension characteristic.

### 3.27.2 Usage

During real-time execution, this array is not recomputed. The size of the array in the type definition for MAX\_COS\_COEFF determines the number of elements of the array to be used in the polynomial evaluation.

# 3.27.2.1 Algorithm

Tow\_burn\_turn\_coeff is used to compute the cosine of the maximum allowed turn angle in each axis for the tow missile during powered flight [burn] using the CSU missile\_util\_cos\_coeff, and called by the CSU missile\_tow\_fly. The CSU missile\_util\_cos\_coeff uses the Newton-Raphson method to evaluate the polynomial with inputs of missile pointer, coefficient array, and time.

```
/*/

* Find the current missile speed and the cosines of the maximum

* allowed turn angles in each direction. The equations used are

* different before and after motor burnout.

/*/

if (time < TOW_BURNOUT_TIME)

{

mptr->speed = mptr->init_speed +
    (speed_factor *
```

#### 3.28 Tow\_coast\_turn\_coeff

The tow\_coast\_turn\_coeff two-dimensional data array consists of the coefficients for three polynomial equations [sidewards, upwards, and downwards movement] defining the TOW missile maximum cosine of turn while unpowered with respect to time in the form using the Newton-Raphson method.

#### 3.28.1 Initialization

The tow\_coast\_turn\_coeff array is initialized during execution of the CSU missile\_tow\_init, called by CSC weapons\_init. Execution of the CSU missile\_tow\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.28. - TOW Missile Coast Turn Coefficients Data Array for a summary of the array data.

The array has a maximum size of 3 by 4 elements.

```
-5.995375e-9, /* a_2 - cos(rad)/tick**3 */
1.162225e-11 /* a_3 - cos(rad)/tick**4 */
},

{
    /* Upwards turn. */
    0.9998498495, /* a_0 - cos(rad)/tick */
    1.657779e-6, /* a_1 - cos(rad)/tick**2 */
    -8.231861e-9, /* a_2 - cos(rad)/tick**3 */
    1.381832e-11 /* a_3 - cos(rad)/tick**4 */
},

{
    /* Downwards turn. */
    0.9999714014, /* a_0 - cos(rad)/tick */
    3.382077e-7, /* a_1 - cos(rad)/tick**2 */
    -1.601259e-9, /* a_2 - cos(rad)/tick**3 */
    2.623014e-12 /* a_3 - cos(rad)/tick**4 */
}
};;
```

Changing the size of the array requires a recompile because of the hard coded multi-dimension characteristic.

### 3.28.2 Usage

During real-time execution, this array is not recomputed. The size of the array in the type definition for MAX\_COS\_COEFF determines the number of elements of the array to be used in the polynomial evaluation.

# 3.28.2.1 Algorithm

Tow\_coast\_turn\_coeff is used to compute the cosine of the maximum allowed turn angle in each axis for the tow missile during unpowered flight [coast] using the CSU missile\_util\_cos\_coeff, and called by the CSU missile\_tow\_fly. The CSU missile\_util\_cos\_coeff uses the Newton-Raphson method to evaluate the polynomial with inputs of missile pointer, coefficient array, and time.

```
/*/

* Find the current missile speed and the cosines of the maximum

* allowed turn angles in each direction. The equations used are

* different before and after motor burnout.

/*/

if (time < TOW_BURNOUT_TIME)

{
```

### 3.29 Adat\_miss\_char

The adat\_miss\_char array consists of characteristics and parameters describing an ADAT missile system and its performance constraints.

# 3.29.1 ADAT\_BURNOUT\_TIME

ADAT\_BURNOUT\_TIME is a constant defining the time of powered flight for the adat missile in ticks.

### 3.29.1.1 Initialization

The constant is initialized during execution of the CSU missile\_adat\_init, called by CSC weapons\_init. Execution of the CSU missile\_adat\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.29. - ADAT Missile Characteristics Data Array for a summary of the constants data.

```
#define ADAT_BURNOUT_TIME adat_miss_char[ 0]
```

### 3.29.1.2 Usage

During real-time execution, this constant is not recomputed.

### 3.29.1.2.1 Algorithm

ADAT\_BURNOUT\_TIME is used to control computation of the missile flyout speed by a call to the CSC missile\_adat\_fly.

```
/*/
* Find the current missile speed and the cosines of the maximum
* allowed turn angles in each direction. The equations used are different
* before and after motor burnout.
  if (time < ADAT_BURNOUT_TIME)
     mptr->speed = missile_util_eval_poly (ADAT_BURN_SPEED_DEG,
         adat_burn_speed_coeff, time) + mptr->init_speed;
     mptr->cos_max_turn[0] = missile_util_eval_poly (
          ADAT_BURN_TURN_DEG, adat_burn_turn_coeff, time);
 }
 else
     mptr->speed = missile_util_eval_poly (ADAT_COAST_SPEED_DEG,
         adat_coast_speed_coeff, time) + mptr->init_speed;
     mptr->cos_max_turn[0] = missile_util_eval_poly (
          ADAT_COAST_TURN_DEG, adat_coast_turn_coeff, time);
 }
```

See APPENDIX E for a complete source code listing.

# 3.29.2 ADAT\_MAX\_FLIGHT\_TIME

ADAT\_MAX\_FLIGHT\_TIME is a constant defining the maximum flight time for the adat missile in ticks.

#### 3.29.2.1 Initialization

The constant is initialized during execution of the CSU missile\_adat\_init, called by CSC weapons\_init. Execution of the CSU missile\_adat\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.29. - ADAT Missile Characteristics Data Array for a summary of the constants data.

#define ADAT\_MAX\_FLIGHT\_TIME adat\_miss\_char[ 1]

#### 3.29.2.2 Usage

During real-time execution, this constant is not recomputed.

### 3.29.2.2.1 Algorithm

ADAT\_MAX\_FLIGHT\_TIME is used to initialize the maximum flight time for an individual adat missile by a call to the CSU missile\_adat\_init.

```
for (i = 0; i < num_missiles; i++)
{
   adat_array[i].mptr.state = ADAT_FREE;
   adat_array[i].mptr.max_flight_time = ADAT_MAX_FLIGHT_TIME;
   adat_array[i].mptr.max_turn_directions = 1;
}</pre>
```

See APPENDIX E for a complete source code listing.

# 3.29.3 INVEST\_DIST\_SQ

INVEST\_DIST\_SQ is a constant defining the area at a maximum speed of less than 100 meters/second.

#### 3.29.3.1 Initialization

The constant is initialized during execution of the CSU missile\_adat\_init, called by CSC weapons\_init. Execution of the CSU missile\_adat\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.29. - ADAT Missile Characteristics Data Array for a summary of the constants data.

```
#define INVEST_DIST_SQ adat_miss_char[ 2]
```

### 3.29.3.2 Usage

During real-time execution, this constant is not recomputed.

# 3.29.3.2.1 Algorithm

INVEST\_DIST\_SQ is used to compute detonation of the proximity fuze by a call to the CSU missile\_fuze\_prox in the CSU missle\_adat\_fly.

```
/*/

* If the missile successfully flew, process the proximity fuze.

/*/

missile_fuze_prox (mptr, MSL_TYPE_MISSILE, aptr->target_flag,
&(aptr->target_vehicle_id), &(aptr->pptr), veh_list,
INVEST_DIST_SQ, aptr->fuze_dist_sq);
```

# 3.29.4 HELO\_FUZE\_DIST\_SQ

HELO\_FUZE\_DIST\_SQ is a constant defining the square of the radius of the cylinder describing the proximity fuze area for a target setting of type HELO.

#### 3.29.4.1 Initialization

The constant is initialized during execution of the CSU missile\_adat\_init, called by CSC weapons\_init. Execution of the CSU missile\_adat\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.29. - ADAT Missile Characteristics Data Array for a summary of the constants data.

```
#define HELO_FUZE_DIST_SQ adat_miss_char[ 3]
```

#### 3.29.4.2 Usage

During real-time execution, this constant is not recomputed.

# 3.29.4.2.1 Algorithm

HELO\_FUZE\_DIST\_SQ is used to compute the fuze distance for the missile target setting by a call to the CSC missile\_adat\_fire.

```
/*/
* Set fuze distance and fuze target according to missile target
* setting. Set network variables.
/*/
   switch (target_type)
   {
      case ADAT_TGT_GND:
        aptr->fuze_dist_sq = 0.0;
        aptr->target_flag = PROX_FUZE_ON_NO_VEH;
```

```
break:
case ADAT TGT HELO:
   aptr->fuze_dist_sq = HELO_FUZE_DIST_SQ;
   if (aptr->target_vehicle id.vehicle == vehicleIrrelevant)
     aptr->target_flag = PROX_FUZE_ON_ALL_VEH;
  else
     aptr->target_flag = PROX_FUZE_ON_ONE_VEH;
 break:
case ADAT TGT AIR:
   aptr->fuze_dist_sq = AIR FUZE DIST SQ;
   if (aptr->target_vehicle_id.vehicle == vehicleIrrelevant)
     aptr->target_flag = PROX FUZE ON ALL VEH;
     aptr->target_flag = PROX_FUZE_ON_ONE_VEH;
 break;
default:
  aptr->fuze\_dist\_sq = 0.0;
   aptr->target_flag = PROX_FUZE_ON_NO_VEH;
  printf ("MISS_ADAT: Unknown target type %d\n", target_type);
 break;
```

The fuze\_dist\_sq is used to compute the proximity fuze by a call to the CSU missile\_fuze\_prox in the CSU missile\_adat\_fly.

See APPENDIX E for a complete source code listing.

### 3.29.5 AIR\_FUZE\_DIST\_SQ

AIR\_FUZE\_DIST\_SQ is a constant defining the square of the radius of the cylinder describing the proximity fuze area for a target setting of type AIR.

#### 3.29.5.1 Initialization

The constant is initialized during execution of the CSU missile\_adat\_init, called by CSC weapons\_init. Execution of the CSU missile\_adat\_init is

normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.29. - ADAT Missile Characteristics Data Array for a summary of the constants data.

```
#define AIR_FUZE_DIST_SQ adat_miss_char[ 4]
```

### 3.29.5.2 Usage

During real-time execution, this constant is not recomputed.

### 3.29.5.2.1 Algorithm

AIR\_FUZE\_DIST\_SQ is used to compute the fuze distance for the missile target setting by a call to the CSC missile\_adat\_fire.

```
/*/
† Set fuze distance and fuze target according to missile target
* setting. Set network variables.
  switch (target_type)
  case ADAT_TGT_GND:
    aptr->fuze_dist_sq = 0.0;
     aptr->target_flag = PROX_FUZE_ON_NO_VEH;
    break;
  case ADAT_TGT_HELO:
     aptr->fuze_dist_sq = HELO_FUZE_DIST_SQ;
     if (aptr->target_vehicle_id.vehicle == vehicleIrrelevant)
       aptr->target_flag = PROX_FUZE_ON_ALL_VEH;
    else
       aptr->target_flag = PROX_FUZE_ON_ONE_VEH;
    break:
   case ADAT_TGT_AIR:
     aptr->fuze_dist_sq = AIR_FUZE_DIST_SQ;
     if (aptr->target_vehicle_id.vehicle == vehicleIrrelevant)
       aptr->target_flag = PROX_FUZE_ON_ALL_VEH;
       aptr->target_flag = PROX_FUZE_ON_ONE_VEH;
    break;
```

```
default:
    aptr->fuze_dist_sq = 0.0;
    aptr->target_flag = PROX_FUZE_ON_NO_VEH;
    printf ("MISS_ADAT: Unknown target type %d\n", target_type);
    break;
}
```

The fuze\_dist\_sq is used to compute the proximity fuze by a call to the CSU missile\_fuze\_prox in the CSU missile\_adat\_fly.

See APPENDIX E for a complete source code listing.

# 3.29.6 ADAT\_TEMP\_BIAS\_TIME

ADAT\_TEMP\_BIAS\_TIME is a constant defining the time of temporal bias for the adat missile in ticks.

#### 3.29.6.1 Initialization

The constant is initialized during execution of the CSU missile\_adat\_init, called by CSC weapons\_init. Execution of the CSU missile\_adat\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.29. - ADAT Missile Characteristics Data Array for a summary of the constants data.

```
#define ADAT_TEMP_BIAS_TIME adat_miss_char[ 5]
```

# 3.29.6.2 Usage

During real-time execution, this constant is not recomputed.

# 3.29.6.2.1 Algorithm

ADAT\_TEMP\_BIAS\_TIME is used to compute the bias for the adat missile by a call to the CSC missile\_adat\_fly.

```
* Find the target point, etc.
 if ((mptr->state == ADAT_GUIDE) | | (mptr->state == ADAT_CLOSE))
     if ((time < ADAT_TEMP_BIAS_TIME) && (mptr->state ==
                                                  ADAT_GUIDE))
        bias = missile_util_eval_poly (ADAT_TEMP_BIAS_DEG,
           adat_temp_bias_coeff, time);
     if (((tube / 2) * 2) == tube)
          missile_target_los_bias (mptr, sight_location,
             loc sight_to_world, -bias, bias);
      else
          missile_target_los_bias (mptr, sight_location,
             loc_sight_to_world, bias, bias);
   }
   else
        missile_target_los (mptr, sight_location, loc_sight_to_world);
  else if (mptr->state == ADAT_UNGUIDE)
     missile_target_unguided (mptr);
 else
     printf ("MISSILE_ADAT: disallowed missile state %d\n", mptr->state);
```

### 3.29.7 CLOSE\_RANGE

CLOSE\_RANGE

#### 3.29.7.1 Initialization

The constant is initialized during execution of the CSU missile\_adat\_init, called by CSC weapons\_init. Execution of the CSU missile\_adat\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.29. - ADAT Missile Characteristics Data Array for a summary of the constants data.

#define CLOSE\_RANGE adat\_miss\_char[6]

#### 3.29.7.2 Usage

During real-time execution, this constant is not recomputed.

# 3.29.7.2.1 Algorithm

CLOSE\_RANGE is used to control the initial orientation of the adat missile by a call to the CSC missile\_adat\_fire.

```
* Set the initial time, location, orientation, and speed of the generic
* missile.
/*/
 mptr->time = 0.0;
  vec_copy (launch_point, mptr->location);
  if (range_to_intercept < CLOSE_RANGE)
     mat_copy (loc_sight_to_world, mptr->orientation);
 else
, {
   if (((tube / 2) * 2) == tube)
         mat_mat_mul (tube_C_sight_left, loc_sight_to_world,
           mptr->orientation);
    else
         mat_mat_mul(tube_C_sight_right, loc_sight_to_world,
           mptr->orientation);
   mptr->speed = missile_util_eval_poly (ADAT_BURN_SPEED_DEG,
       adat_burn_speed_coeff, 0.0) + launch_speed;
  mptr->init_speed = launch_speed;
```

CLOSE\_RANGE is used to control the state of guidance for the adat missile by a call to the CSC missile\_adat\_fire.

```
/*/
* If all was successful, put any flying missiles in an unguided state
* and put this missile in a guided state.
/*/
for (i = 0; i < num_adats; i++)
{
   if ((adat_array[i].mptr.state == ADAT_GUIDE) | |
        (adat_array[i].mptr.state == ADAT_CLOSE))
        adat_array[i].mptr.state = ADAT_UNGUIDE;
}</pre>
```

```
if (range_to_intercept < CLOSE_RANGE)
    mptr->state = ADAT_CLOSE;
else
    mptr->state = ADAT_GUIDE;
```

# 3.30 Adat\_miss\_poly\_deg

The adat\_miss\_poly\_deg array consists of values of the degree of each polynomial equation used to compute the burn speed, the coast speed, maximum cosines of turns while powered, maximum cosines of turns while unpowered, and temporal bias for the ADAT missile.

# 3.30.1 ADAT\_BURN\_SPEED\_DEG

ADAT\_BURN\_SPEED\_DEG is a constant defining the polynomial degree for ADAT missile burn speed coefficient data array. ADAT\_BURN\_SPEED\_DEG is the first element of the adat\_miss\_poly\_deg.

### 3.30.1.1 Initialization

The constant is initialized during execution of the CSU missile\_adat\_init, called by CSC weapons\_init. Execution of the CSU missile\_adat\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.30. - ADAT Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define ADAT_BURN_SPEED_DEG adat_miss_poly_deg[ 0]
```

# 3.30.1.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for ADAT\_BURN\_SPEED\_DEG is 9, especially, the declared size of the adat\_burn\_speed\_coeff is 10.

# 3.30.1.2.1 Algorithm

ADAT\_BURN\_SPEED\_DEG is used to compute the ADAT missile speed at launch using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_adat\_fire. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
* Set the initial time, location, orientation, and speed of the generic
* missile.
 mptr->time = 0.0;
  vec_copy (launch_point, mptr->location);
  if (range_to_intercept < CLOSE_RANGE)</pre>
     mat_copy (loc_sight_to_world, mptr->orientation);
 else
   if (((tube / 2) * 2) == tube)
         mat_mat_mul (tube_C_sight_left, loc_sight_to_world,
           mptr->orientation);
   else
         mat_mat_mul(tube_C_sight_right, loc_sight_to_world,
           mptr->orientation);
   mptr->speed = missile_util_eval_poly (ADAT_BURN_SPEED_DEG,
       adat burn speed_coeff, 0.0) + launch_speed;
  mptr->init_speed = launch_speed;
```

ADAT\_BURN\_SPEED\_DEG is used to compute the ADAT missile speed during powered flight [burn] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_adat\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/

* Find the current missile speed and the cosines of the maximum allowed

* turn angles in each direction. The equations used are different before and

* after motor burnout.

/*/

if (time < ADAT_BURNOUT_TIME)

{

mptr->speed = missile_util_eval_poly (ADAT_BURN_SPEED_DEG,
adat_burn_speed_coeff, time) + mptr->init_speed;
mptr->cos_max_turn[0] = missile_util_eval_poly(
ADAT_BURN_TURN_DEG,
adat_burn_turn_coeff, time);
}
else
{
```

### 3.30.2 ADAT\_COAST\_SPEED\_DEG

ADAT\_COAST\_SPEED\_DEG is a constant defining the polynomial degree for adat missile coast speed coefficient data array. ADAT\_COAST\_SPEED\_DEG is the second element of the adat\_miss\_poly\_deg.

#### 3.30.2.1 Initialization

The constant is initialized during execution of the CSU missile\_adat\_init, called by CSC weapons\_init. Execution of the CSU missile\_adat\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.30. - ADAT Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define ADAT_COAST_SPEED_DEG adat_miss_poly_deg[ 1]
```

#### 3.30.2.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for ADAT\_COAST\_SPEED\_DEG is 9, especially, the declared size of the adat\_coast\_speed\_coeff is 10.

### 3.30.2.2.1 Algorithm

ADAT\_COAST\_SPEED\_DEG is used to compute the ADAT missile speed during unpowered flight [coast] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_adat\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
* Find the current missile speed and the cosines of the maximum allowed
* turn angles in each direction. The equations used are different before and
* after motor burnout.
/*/
  if (time < ADAT_BURNOUT_TIME)
     mptr->speed = missile_util_eval_poly (ADAT_BURN_SPEED_DEG,
         adat_burn_speed_coeff, time) + mptr->init_speed;
     mptr->cos_max_turn[0] = missile_util_eval_poly(
                       ADAT_BURN_TURN_DEG,
                       adat_burn_turn_coeff, time);
 }
  else
     mptr->speed = missile_util_eval_poly (ADAT_COAST_SPEED_DEG,
         adat_coast_speed_coeff, time) + mptr->init_speed;
     mptr->cos_max_turn[0] = missile_util_eval_poly(
                       ADAT_COAST_TURN_DEG,
                        adat_coast_turn_coeff, time);
  }
```

# 3.30.3 ADAT\_BURN\_TURN\_DEG

ADAT\_BURN\_TURN\_DEG is a constant defining the polynomial degree for the adat missile maximum cosine of turn angle, burn turn coefficient data array. ADAT\_BURN\_TURN\_DEG is the third element of the adat\_miss\_poly\_deg.

#### 3.30.3.1 Initialization

The constant is initialized during execution of the CSU missile\_adat\_init, called by CSC weapons\_init. Execution of the CSU missile\_adat\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.30. - ADAT Missile Polynomial Degree Data Array for a summary of the constants data.

#define ADAT\_BURN\_TURN\_DEG adat\_miss\_poly\_deg[ 2]

### 3.30.3.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for ADAT\_BURN\_TURN\_DEG is 9, especially, the declared size of the adat\_burn\_turn\_coeff is 10.

### 3.30.3.2.1 Algorithm

ADAT\_BURN\_TURN\_DEG is used to compute cosine of the maximum allowed turn angle for the ADAT missile during powered flight [burn] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_adat\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
* Find the current missile speed and the cosines of the maximum allowed
* turn angles in each direction. The equations used are different before and
*, after motor burnout.
  if (time < ADAT_BURNOUT_TIME)
     mptr->speed = missile_util_eval_poly (ADAT_BURN_SPEED_DEG,
         adat_burn_speed_coeff, time) + mptr->init_speed;
     mptr->cos max turn[0] = missile_util_eval_poly(
                       ADAT_BURN_TURN_DEG,
                       adat_burn_turn_coeff, time);
 }
 else
     mptr->speed = missile_util_eval_poly (ADAT_COAST_SPEED_DEG,
         adat_coast_speed_coeff, time) + mptr->init_speed;
     mptr->cos_max_turn[0] = missile_util_eval_poly(
                       ADAT COAST_TURN_DEG,
                        adat_coast_turn_coeff, time);
 }
```

See APPENDIX E for a complete source code listing.

# 3.30.4 ADAT\_COAST\_TURN\_DEG

ADAT\_COAST\_TURN\_DEG is a constant defining the polynomial degree for the adat missile maximum cosine of turn angle, coast turn coefficient data

array. ADAT\_COAST\_TURN\_DEG is the fourth element of the adat\_miss\_poly\_deg.

#### 3.30.4.1 Initialization

The constant is initialized during execution of the CSU missile\_adat\_init, called by CSC weapons\_init. Execution of the CSU missile\_adat\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.30. - ADAT Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define ADAT_COAST_TURN_DEG adat_miss_poly_deg[ 3]
```

#### 3.30.4.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for ADAT\_COAST\_TURN\_DEG is 9, especially, the declared size of the adat\_coast\_turn\_coeff is 10.

### 3.30.4.2.1 Algorithm

ADAT\_COAST\_TURN\_DEG is used to compute the cosine of the maximum allowed turn angle for the ADAT missile during unpowered flight [coast] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_adat\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/

* Find the current missile speed and the cosines of the maximum allowed

* turn angles in each direction. The equations used are different before and

* after motor burnout.

/*/

if (time < ADAT_BURNOUT_TIME)

{

mptr->speed = missile_util_eval_poly (ADAT_BURN_SPEED_DEG,
adat_burn_speed_coeff, time) + mptr->init_speed;
mptr->cos_max_turn[0] = missile_util_eval_poly(
ADAT_BURN_TURN_DEG,
adat_burn_turn_coeff, time);
}
else
{
```

# 3.30.5 ADAT\_TEMP\_BIAS\_DEG

ADAT\_TEMP\_BIAS\_DEG is a constant defining the polynomial degree for the adat missile temporal bias coefficient data array

#### 3.30.5.1 Initialization

The constant is initialized during execution of the CSU missile\_adat\_init, called by CSC weapons\_init. Execution of the CSU missile\_adat\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.30. - ADAT Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define ADAT_TEMP_BIAS_DEG adat_miss_poly_deg[ 4]
```

### 3.30.5.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for ADAT\_TEMP\_BIAS\_DEG is 9, especially, the declared size of the adat\_coast\_turn\_coeff is 10.

# 3.30.5.2.1 Algorithm

ADAT\_TEMP\_BIAS\_DEG is used to compute the temporal bias applied to the target location using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_adat\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
* Find the target point, etc.
/*/
if ((mptr->state == ADAT_GUIDE) || (mptr->state == ADAT_CLOSE))
{
```

```
if ((time < ADAT_TEMP_BIAS_TIME) && (mptr->state ==
                                                 ADAT_GUIDE))
      bias = missile_util eval poly (ADAT TEMP_BIAS_DEG,
          adat_temp_bias_coeff, time);
    if (((tube / 2) * 2) == tube)
        missile_target_los_bias (mptr, sight_location,
           loc sight_to_world, -bias, bias);
    else
        missile_target_los_bias (mptr, sight_location,
           loc_sight_to_world, bias, bias);
  }
  else
      missile_target_los (mptr, sight_location, loc_sight_to_world);
else if (mptr->state == ADAT_UNGUIDE)
   missile_target_unguided (mptr);
else
   printf ("MISSILE_ADAT: disallowed missile state %d\n", mptr->state);
```

### 3.31 Adat\_burn\_speed\_coeff

The adat\_burn\_speed array consists of the coefficients for a polynomial equation defining the ADAT missile burn speed with respect to time in the form using the Newton-Raphson method.

#### 3.31.1 Initialization

The adat\_burn\_speed array is initialized during execution of the CSU missile\_adat\_init, called by CSC weapons\_init. Execution of the CSU missile\_adat\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.31. - ADAT Missile Burn Speed Coefficient Data Array for a summary of the array data.

The following is the default declaration.

```
/*/
* Coefficients for the speed polynomial before motor burnout.
/*/
static REAL adat_burn_speed_coeff[10] =
{
```

The array has a maximum size of 10 elements.

#### 3.31.2 Usage

During real-time execution, this array is not recomputed. ADAT\_BURN\_SPEED\_DEG determines the number of elements of the array to be used in the polynomial evaluation.

# 3.31.2.1 Algorithm

The adat\_burn\_speed\_coeff array is used to initialize the tube to sight transformation matrices by a call to the CSU missile\_adat\_init.

```
* Initialize the tube to sight transformation matrices.
  mag = sqrt (adat_burn_speed_coeff[0] * adat_burn_speed_coeff[0] +
       2.0 * adat_temp_bias_coeff[0] * adat_temp_bias_coeff[0]);
  tube_C_sight_right[1][0] = adat_temp_bias_coeff[0] / mag;
  tube_C_sight_right[1][1] = adat_burn_speed_coeff[0] / mag;
  tube_C_sight_right[1][2] = adat_temp_bias_coeff[0] / mag;
  mag = sqrt (tube_C_sight_right[1][0] * tube_C_sight_right[1][0] +
       tube_C_sight_right[1][1] * tube_C_sight_right[1][1]);
  tube_C_sight_right[0][0] = tube_C_sight_right[1][1] / mag;
  tube_C_sight_right[0][1] = -tube_C_sight_right[1][0] / mag;
  tube_C_sight_right[0][2] = 0.0;
  tube_C_sight_right[2][0] = tube_C_sight_right[1][2] *
       tube C_sight_right[0][1];
  tube_C_sight_right[2][1] = -tube_C_sight_right[1][2] *
       tube_C_sight_right[0][0];
  tube_C_sight_right[2][2] = mag;
   mat_copy (tube_C_sight_right, tube_C_sight_left);
  tube_C_sight_left[0][1] = -tube_C_sight_left[0][1];
```

```
tube_C_sight_left[1][0] = -tube_C_sight_left[1][0];
tube_C_sight_left[2][0] = -tube_C_sight_left[2][0];
```

The adat\_burn\_speed\_coeff array is used to compute the initial speed at launch of the ADAT missile using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_adat\_fire. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
mptr->speed = missile_util_eval_poly (ADAT_BURN_SPEED_DEG,
    adat_burn_speed_coeff, 0.0) + launch_speed;
```

The adat\_burn\_speed\_coeff array is used to compute the ADAT missile speed during powered flight [burn] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_adat\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
* Find the current missile speed and the cosines of the maximum allowed
turn
* angles in each direction. The equations used are different before and
* after motor burnout.
/*/
  if (time < ADAT_BURNOUT_TIME)
     mptr->speed = missile_util_eval_poly (ADAT_BURN_SPEED_DEG,
         adat_burn_speed_coeff, time) + mptr->init_speed;
     mptr->cos_max_turn[0] = missile_util_eval_poly (
                              ADAT_BURN_TURN_DEG,
                              adat_burn_turn_coeff, time);
 }
  else
     mptr->speed = missile_util_eval_poly (ADAT_COAST_SPEED_DEG,
         adat_coast_speed_coeff, time) + mptr->init_speed;
     mptr->cos_max_turn[0] = missile_util_eval_poly (
                              ADAT_COAST_TURN_DEG,
                              adat_coast_turn_coeff, time);
  }
```

#### 3.32 Adat\_coast\_speed\_coeff

This data array consists of the coefficients for a polynomial equation defining the ADAT missile coast speed with respect to time in the form using the Newton-Raphson method.

#### 3.32.1 Initialization

The adat\_coast\_speed array is initialized during execution of the CSU missile\_adat\_init, called by CSC weapons\_init. Execution of the CSU missile\_adat\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.32. - ADAT Missile Coast Speed Coefficient Data Array for a summary of the array data.

The following is the default declaration.

```
/*/
* Coefficients for the speed polynomial after motor burnout.
/*/
static REAL adat_coast_speed_coeff[10] =
 105.52162,
                    /* a 0 - m/tick */
                   /* a_1 - m/tick**2 */
 -1.0157285,
 5.6124330e-3, /* a_2 - m/tick**3 */
-1.6262608e-5, /* a_3 - m/tick**4 */
  1.8991982e-8,
                   /* a 4 - m/tick**5 */
  0.0,
  0.0,
  0.0,
  0.0,
  0.0
};
```

The array has a maximum size of 10 elements.

### 3.32.2 Usage

During real-time execution, this array is not recomputed. ADAT\_COAST\_SPEED\_DEG determines the number of elements of the array to be used in the polynomial evaluation.

#### 3.32.2.1 Algorithm

The adat\_coast\_speed\_coeff array is used to compute the ADAT missile speed during unpowered flight [coast] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_adat\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
Find the current missile speed and the cosines of the maximum allowed
* angles in each direction. The equations used are different before and
* after motor burnout.
  if (time < ADAT_BURNOUT_TIME)
     mptr->speed = missile_util_eval_poly (ADAT_BURN_SPEED_DEG,
         adat burn_speed_coeff, time) + mptr->init_speed;
    mptr->cos_max_turn[0] = missile_util_eval_poly (
                             ADAT_BURN_TURN_DEG,
                             adat burn_turn_coeff, time);
 }
 else
     mptr->speed = missile_util_eval_poly (ADAT_COAST_SPEED_DEG,
         adat coast speed_coeff, time) + mptr->init_speed;
     mptr->cos_max_turn[0] = missile_util_eval_poly (
                             ADAT_COAST_TURN_DEG,
                             adat coast_turn_coeff, time);
 }
```

See APPENDIX E for a complete source code listing.

# 3.33 Adat\_burn\_turn\_coeff

The adat\_burn\_turn\_coeff array consists of the coefficients for a polynomial equation defining the ADAT missile maximum cosine of turn while powered with respect to time in the form using the Newton-Raphson method.

#### 3.33.1 Initialization

The adat\_burn\_turn array is initialized during execution of the CSU missile\_adat\_init, called by CSC weapons\_init. Execution of the CSU missile\_adat\_init is normally done only once during CSCI initialization and

is performed sequentially. See TABLE 5.1.33. - ADAT Missile Burn Turn Coefficient Data Array for a summary of the array data.

The following is the default declaration.

The array has a maximum size of 10 elements.

### 3.33.2 Usage

During real-time execution, this array is not recomputed. ADAT\_BURN\_TURN\_DEG determines the number of elements of the array to be used in the polynomial evaluation.

# 3.33.2.1 Algorithm

The adat\_burn\_turn\_coeff array is used to compute the cosine of the maximum allowed turn angle of the ADAT missile speed during powered flight [burn] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_adat\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/
* Find the current missile speed and the cosines of the maximum allowed turn
```

### 3.34 Adat\_coast\_turn\_coeff

This data array consists of the coefficients for a polynomial equation defining the ADAT missile maximum cosine of turn while unpowered with respect to time in the form using the Newton-Raphson method.

#### 3.34.1 Initialization

The adat\_coast\_turn array is initialized during execution of the CSU missile\_adat\_init, called by CSC weapons\_init. Execution of the CSU missile\_adat\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.34. - ADAT Missile Coast Turn Coefficient Data Array for a summary of the array data.

The following is the default declaration.

```
/*/

* Coefficients for the cosine of max turn polynomial after motor burnout.
/*/

static REAL adat_coast_turn_coeff[10] =
{
```

The array has a maximum size of 10 elements.

#### 3.34.2 Usage

During real-time execution, this array is not recomputed. ADAT\_COAST\_TURN\_DEG determines the number of elements of the array to be used in the polynomial evaluation.

### 3.34.2.1 Algorithm

The adat\_coast\_turn\_coeff array is used to compute the cosine of the maximum allowed turn angle of the ADAT missile speed during unpowered flight [coast] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_adat\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/

* Find the current missile speed and the cosines of the maximum allowed turn

* angles in each direction. The equations used are different before and

* after motor burnout.

/*/

if (time < ADAT_BURNOUT_TIME)

{

mptr->speed = missile_util_eval_poly (ADAT_BURN_SPEED_DEG,
adat_burn_speed_coeff, time) + mptr->init_speed;
mptr->cos_max_turn[0] = missile_util_eval_poly (
ADAT_BURN_TURN_DEG,
adat_burn_turn_coeff, time);
}
else
```

# 3.35 Adat\_temp\_bias\_coeff

The adat\_temp\_bias\_coeff array consists of the coefficients for a polynomial equation defining the ADAT missile temporal bias with respect to time in the form using the Newton-Raphson method.

### 3.35.1 Initialization

The adat\_temp\_bias\_coeff array is initialized during execution of the CSU missile\_adat\_init, called by CSC weapons\_init. Execution of the CSU missile\_adat\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.35. - ADAT Missile Temporal Bias Coefficient Data Array for a summary of the array data.

The following is the default declaration.

The array has a maximum size of 10 elements.

### 3.35.2 Usage

During real-time execution, this array is not recomputed. ADAT\_TEMP\_BIAS\_DEG determines the number of elements of the array to be used in the polynomial evaluation.

# 3.35.2.1 Algorithm

The adat\_temp\_bias\_coeff array is used to initialize the tube to sight transformation matrices by a call to the CSU missile\_adat\_init.

```
* Initialize the tube to sight transformation matrices.
  mag = sqrt (adat_burn_speed_coeff[0] * adat_burn_speed_coeff[0] +
       2.0 * adat_temp_bias_coeff[0] * adat_temp_bias_coeff[0]);
  tube_C_sight_right[1][0] = adat_temp_bias_coeff[0] / mag;
  tube_C_sight_right[1][1] = adat_burn_speed_coeff[0] / mag;
  tube_C_sight_right[1][2] = adat_temp_bias_coeff[0] / mag;
  mag = sqrt (tube_C_sight_right[1][0] * tube_C_sight_right[1][0] +
       tube_C_sight_right[1][1] * tube_C_sight_right[1][1]);
  tube_C_sight_right[0][0] = tube_C_sight_right[1][1] / mag;
  tube_C_sight_right[0][1] = -tube_C_sight_right[1][0] / mag;
  tube_C_{sight_right[0][2]} = 0.0;
  tube_C_sight_right[2][0] = tube_C_sight_right[1][2] *
       tube_C_sight_right[0][1];
  tube_C_sight_right[2][1] = -tube_C_sight_right[1][2] *
       tube_C_sight_right[0][0];
  tube_C_sight_right[2][2] = mag;
   mat_copy (tube_C_sight_right, tube_C_sight_left);
  tube_C_sight_left[0][1] = -tube_C_sight_left[0][1];
  tube_C_sight_left[1][0] = -tube_C_sight_left[1][0];
  tube_C_sight_left[2][0] = -tube_C_sight_left[2][0];
```

The adat\_temp\_bias\_coeff array is used to compute the temporal bias applied to the target location using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_adat\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
* Find the target point, etc.
 if ((mptr->state == ADAT_GUIDE) | | (mptr->state == ADAT_CLOSE))
    if ((time < ADAT_TEMP_BIAS_TIME) && (mptr->state ==
                                                   ADAT_GUIDE))
   {
        bias = missile_util_eval_poly (ADAT_TEMP_BIAS_DEG,
           adat_temp_bias_coeff, time);
     if (((tube / 2) * 2) == tube)
          missile_target_los_bias (mptr, sight_location,
             loc_sight_to_world, -bias, bias);
      else
          missile_target_los_bias (mptr, sight_location,
             loc sight to world, bias, bias);
   }
    else
        missile_target_los (mptr, sight_location, loc_sight_to_world);
  else if (mptr->state == ADAT_UNGUIDE)
     missile_target_unguided (mptr);
 else
     printf ("MISSILE_ADAT: disallowed missile state %d\n", mptr->state);
```

# 3.36 Atgm\_miss\_char

The atgm\_miss\_char array consists of characteristics and parameters describing an ATGM missile system and its performance constraints. The tow missile source code was used as the baseline for the ATGM missile function; many of the ATGM constants, variables, CSCs and CSUs have the same name as in the TOW missile source code.

# 3.36.1 TOW\_BURNOUT\_TIME [for ATGM]

TOW\_BURNOUT\_TIME is a constant defining the time of powered flight for ATGM missile in ticks.

#### 3.36.1.1 Initialization

The constant is initialized during execution of the CSU missile\_atgm\_init, called by CSC weapons\_init. Execution of the CSU missile\_atgm\_init is normally done only once during CSCI initialization and is performed

sequentially. See TABLE 5.1.36. - ATGM Missile Characteristics Data Array for a summary of the constants data.

```
#define TOW_BURNOUT_TIME tow_miss_char[ 0]
```

# 3.36.1.2 Usage

During real-time execution, this constant is not recomputed.

# 3.36.1.2.1 Algorithm

TOW\_BURNOUT\_TIME is used to control computation of the missile flyout speed and the cosines of the maximum allowed turn angles in each direction by a call to the CSC missile\_atgm\_fly.

```
/*/
* Find the current missile speed and the cosines of the maximum
     allowed turn angles in each direction. The equations used are
     different before and after motor burnout.
  if (time < TOW_BURNOUT_TIME)
    mptr->speed = mptr->init_speed +
      (speed_factor *
         missile_util_eval_poly (TOW_BURN_SPEED_DEG,
                     tow_burn_speed_coeff, time));
      missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
  }
  else
     mptr->speed = mptr->init_speed +
       (speed_factor *
         missile_util_eval_poly (TOW_COAST_SPEED_DEG,
                     tow_coast_speed_coeff, time));
      missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
  }
```

See APPENDIX F for a complete source code listing.

# 3.36.2 TOW\_RANGE\_LIMIT\_TIME [for ATGM]

TOW\_RANGE\_LIMIT\_TIME is a constant defining the range limit time for the ATGM missile in ticks; at this point the wire is cut, but the missile is allowed to fly to the maximum flight time.

### 3.36.2.1 Initialization

The constant is initialized during execution of the CSU missile\_atgm\_init, called by CSC weapons\_init. Execution of the CSU missile\_atgm\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.36. - ATGM Missile Characteristics Data Array for a summary of the constants data.

```
#define TOW_RANGE_LIMIT_TIME tow_miss_char[ 1]
```

### 3.36.2.2 Usage

During real-time execution, this constant is not recomputed.

# 3.36.2.2.1 Algorithm

TOW\_RANGE\_LIMIT\_TIME is used to control the wire cut at the maximum range flight time for an individual ATGM missile by a call to the CSU missile\_atgm\_fly.

```
/*/

* If the missile has reached its maximum range (not the maximum distance

* its allowed to fly), cut the wire.

/*/

if ((time > TOW_RANGE_LIMIT_TIME) && !tptr->wire_is_cut)

tptr->wire_is_cut = TRUE;
```

See APPENDIX F for a complete source code listing.

# 3.36.3 TOW\_MAX\_FLIGHT\_TIME [for ATGM]

TOW\_MAX\_FLIGHT\_TIME is a constant defining the maximum flight time for the ATGM missile assumed in ticks; cosine of the maximum turn angle is greater than 1.0 beyond this point.

### 3.36.3.1 Initialization

The constant is initialized during execution of the CSU missile\_atgm\_init, called by CSC weapons\_init. Execution of the CSU missile\_atgm\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.36. - ATGM Missile Characteristics Data Array for a summary of the constants data.

#define TOW\_MAX\_FLIGHT\_TIME tow\_miss\_char[ 2]

# 3.36.3.2 Usage

During real-time execution, this constant is not recomputed.

# 3.36.3.2.1 Algorithm

TOW\_MAX\_FLIGHT\_TIME is used to initialize the maximum flight time for an individual ATGM missile by a call to the CSU missile\_atgm\_init.

tptr->mptr.max\_flight\_time = TOW\_MAX\_FLIGHT\_TIME;

See APPENDIX F for a complete source code listing.

# 3.36.4 ATGM\_TURN\_FACTOR

ATGM\_TURN\_FACTOR is a constant defining the ratio of the ATGM to TOW missile performance in turns; ATGM turn factor for wider turning capability with respect to TOW

### 3.36.4.1 Initialization

The constant is initialized during execution of the CSU missile\_atgm\_init, called by CSC weapons\_init. Execution of the CSU missile\_atgm\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.36. - ATGM Missile Characteristics Data Array for a summary of the constants data.

#define ATGM\_TURN\_FACTOR tow\_miss\_char[3]

#### 3.36.4.2 Usage

During real-time execution, this constant is not recomputed.

### 3.36.4.2.1 Algorithm

ATGM\_TURN\_FACTOR is used to modify the tow burn turn and tow coast turn coefficients for each axis by a call to the CSU missile\_atgm\_init.

See APPENDIX F for a complete source code listing.

# 3.37 Atgm\_miss\_poly\_deg

The atgm\_miss\_poly\_deg array consists of values of the degree of each polynomial equation used to compute the burn speed, the coast speed, maximum cosines of turns while powered, and maximum cosines of turns while unpowered for the ATGM missile. The tow missile source code was used as the baseline for the ATGM missile function; many of the ATGM constants, variables, CSCs and CSUs have the same name as in the TOW missile source code.

# 3.37.1 TOW\_BURN\_SPEED\_DEG [for ATGM]

TOW\_BURN\_SPEED\_DEG is a constant defining the polynomial degree for the ATGM missile burn speed coefficient data array.

#### 3.37.1.1 Initialization

The constant is initialized during execution of the CSU missile\_atgm\_init, called by the CSC weapons\_init. Execution of the CSU missile\_atgm\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.37. - ATGM Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define TOW_BURN_SPEED_DEG tow_miss_poly_deg[0]
```

### 3.37.1.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for TOW\_BURN\_SPEED\_DEG is 4, especially, the declared size of the tow\_burn\_speed\_coeff is 5.

### 3.37.1.2.1 Algorithm

TOW\_BURN\_SPEED\_DEG is used to compute the ATGM missile speed at launch using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_atgm\_fire. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/

* Set the initial time, location, orientation, and speed of the generic

* missile.

/*/

mptr->time = 0.0;

vec_copy (launch_point, mptr->location);

mat_copy (loc_sight_to_world, mptr->orientation);

mptr->speed = launch_speed +

(speed_factor * missile_util_eval_poly (TOW_BURN_SPEED_DEG,
tow_burn_speed_coeff, 0.0));

mptr->init_speed = launch_speed;
```

TOW\_BURN\_SPEED\_DEG is used to compute the ATGM missile speed during powered flight [burn] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_atgm\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/

* Find the current missile speed and the cosines of the maximum allowed

* turn angles in each direction. The equations used are different before and

* after motor burnout.

/*/

if (time < TOW_BURNOUT_TIME)

{

mptr->speed = missile_util_eval_poly (TOW_BURN_SPEED_DEG,
tow_burn_speed_coeff, time) + mptr->init_speed;
missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);

}
else

{

mptr->speed = missile_util_eval_poly (TOW_COAST_SPEED_DEG,
tow_coast_speed_coeff, time) + mptr->init_speed;
missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
}
```

# 3.37.2 TOW\_COAST\_SPEED\_DEG [for ATGM]

TOW\_COAST\_SPEED\_DEG is a constant defining the polynomial degree for ATGM missile coast speed coefficient data array

### 3.37.2.1 Initialization

The constant is initialized during execution of the CSU missile\_atgm\_init, called by the CSC weapons\_init. Execution of the CSU missile\_atgm\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.37. - ATGM Missile Polynomial Degree Data Array for a summary of the array data.

```
#define TOW_COAST_SPEED_DEG tow_miss_poly_deg[1]
```

# 3.37.2.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for TOW\_COAST\_SPEED\_DEG is 4, especially, the declared size of the tow\_burn\_speed\_coeff is 5.

### 3.37.2.2.1 Algorithm

TOW\_COAST\_SPEED\_DEG is used to compute the ATGM missile speed during unpowered flight [coast] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_atgm\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/

* Find the current missile speed and the cosines of the maximum allowed

* turn angles in each direction. The equations used are different before and

* after motor burnout.

/*/

if (time < TOW_BURNOUT_TIME)

{

mptr->speed = missile_util_eval_poly (TOW_BURN_SPEED_DEG,
tow_burn_speed_coeff, time) + mptr->init_speed;
missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
}

else

{

mptr->speed = missile_util_eval_poly (TOW_COAST_SPEED_DEG,
tow_coast_speed_coeff, time) + mptr->init_speed;
missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
}
```

See APPENDIX F for a complete source code listing.

# 3.37.3 TOW\_BURN\_TURN\_DEG [for ATGM]

TOW\_BURN\_TURN\_DEG is a constant defining the polynomial degree for each ATGM missile burn turn coefficient data sub-array of the ATGM missile burn turn coefficient data array structure

#### 3.37.3.1 Initialization

The constant is initialized during execution of the CSU missile\_atgm\_init, called by CSC weapons\_init. Execution of the CSU missile\_atgm\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.37. - ATGM Missile Polynomial Degree Data Array for a summary of the constant data.

```
#define TOW_BURN_TURN_DEG tow_miss_poly_deg[2]
```

### 3.37.3.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for TOW\_BURN\_TURN\_DEG is 1, especially, the declared size of the tow\_burn\_turn\_coeff is 2. Changing this constant requires a recompile because of the hard coded multi-dimension characteristic.

```
* Coefficients for the cosine of max turn polynomials before motor burnout.
* The structure _MAX_COS_COEFF_ is used to store the values for the turn
* sideways, up, and down polynomials along with their order.
/*/
static MAX_COS_COEFF tow_burn_turn_coeff =
              /* Order of the polynomials. */
  1,
  {
              /* Sidewards turn. */
    0.999976868652, /* a_0 - cos(rad)/tick */
    -3.5933955e-7 /* a_1 - cos(rad)/tick**2 */
  },
  {
              /* Upwards turn. */
    0.999960667258, /* a_0 - cos(rad)/tick */
    -3.1492328e-6 /* a_1 - cos(rad)/tick**2 */
  },
              /* Downwards turn. */
    0.999978909989, /* a_0 - cos(rad)/tick */
    -7.8194991e-9 /* a_1 - cos(rad)/tick**2 */
};
```

# 3.24.3.2.1 Algorithm

TOW\_BURN\_TURN\_DEG is hard coded by type definition of MAX\_COS\_COEFF and is used to compute the cosine of the maximum allowed turn angle in each axis for the ATGM missile during powered flight [burn] using the CSU missile\_util\_cos\_coeff, and called by the CSU missile\_atgm\_fly. The CSU missile\_util\_cos\_coeff uses the Newton-

Raphson method to evaluate the polynomial with inputs of missile pointer, coefficient array, and time.

```
/*/

* Find the current missile speed and the cosines of the maximum allowed

* turn angles in each direction. The equations used are different before and

* after motor burnout.

/*/

if (time < TOW_BURNOUT_TIME)

{

mptr->speed = missile_util_eval_poly (TOW_BURN_SPEED_DEG,
tow_burn_speed_coeff, time) + mptr->init_speed;
missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
}

else

{

mptr->speed = missile_util_eval_poly (TOW_COAST_SPEED_DEG,
tow_coast_speed_coeff, time) + mptr->init_speed;
missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
}
```

See APPENDIX F for a complete source code listing.

# 3.37.4 TOW\_COAST\_TURN\_DEG [for ATGM]

TOW\_COAST\_TURN\_DEG is a constant defining the polynomial degree for each ATGM missile coast turn coefficient data sub-array of the ATGM missile coast turn coefficient data array structure

#### 3.37.4.1 Initialization

The constant is initialized during execution of the CSU missile\_atgm\_init, called by CSC weapons\_init. Execution of the CSU missile\_atgm\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.37. - ATGM Missile Polynomial Degree Data Array for a summary of the array data.

#define TOW\_COAST\_TURN\_DEG tow\_miss\_poly\_deg[3]

#### 3.37.4.2 Usage

During real-time execution, this constant is not recomputed. The maximum value in each axis for TOW\_COAST\_TURN\_DEG is 3, especially, the declared size of the tow\_coast\_turn\_coeff is 4. Changing this constant requires a recompile because of the hard coded multi-dimension characteristic.

```
* Coefficients for the cosine of max turn polynomials after motor burnout.
static MAX_COS_COEFF tow_coast_turn_coeff =
              /* Order of the polynomials. */
  3,
             /* Sidewards turn. */
    0.99995112518, /* a_0 - cos(rad)/tick */
    8.96333e-7, /* a_1 - cos(rad)/tick**2 */
    -5.995375e-9, /* a_2 - cos(rad)/tick**3 */
    1.162225e-11 /* a_3 - cos(rad)/tick**4 */
  },
             /* Upwards turn. */
    1.657779e-6, /* a_1 - cos(rad)/tick**2 */
    -8.231861e-9, /* a_2 - cos(rad)/tick**3 */
    1.381832e-11 /* a_3 - cos(rad)/tick**4 */
  },
              /* Downwards turn. */
    0.9999714014, /* a_0 - cos(rad)/tick */
    3.382077e-7, /* a_1 - cos(rad)/tick**2 */
    -1.601259e-9, /* a_2 - cos(rad)/tick**3 */
    2.623014e-12 /* a_3 - cos(rad)/tick**4 */
```

# 3.37.4.2.1 Algorithm

TOW\_COAST\_TURN\_DEG is hard coded by type definition of MAX\_COS\_COEFF and is used to compute the cosine of the maximum allowed turn angle in each axis for the atgm missile during unpowered flight

[coast] using the CSU missile\_util\_cos\_coeff, and called by the CSU missile\_atgm\_fly. The CSU missile\_util\_cos\_coeff uses the Newton-Raphson method to evaluate the polynomial with inputs of missile pointer, coefficient array, and time.

```
/*/

* Find the current missile speed and the cosines of the maximum allowed

* turn angles in each direction. The equations used are different before and

* after motor burnout.

/*/

if (time < TOW_BURNOUT_TIME)

{

mptr->speed = missile_util_eval_poly (TOW_BURN_SPEED_DEG,
tow_burn_speed_coeff, time) + mptr->init_speed;
missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
}

else

{

mptr->speed = missile_util_eval_poly (TOW_COAST_SPEED_DEG,
tow_coast_speed_coeff, time) + mptr->init_speed;
missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
}
```

See APPENDIX F for a complete source code listing.

# 3.38 Tow\_burn\_speed\_coeff [for ATGM]

The tow\_burn\_speed\_coeff array consists of the coefficients for a polynomial equation defining the ATGM missile burn speed with respect to time in the form using the Newton-Raphson method. The tow missile source code was used as the baseline for the ATGM missile function; many of the ATGM constants, variables, CSCs and CSUs have the same name as in the TOW missile source code.

#### 3.38.1 Initialization

The tow\_burn\_speed\_coeff array is initialized during execution of the CSU missile\_atgm\_init, called by CSC weapons\_init. Execution of the CSU missile\_atgm\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.38. - ATGM Missile Burn Speed Coefficient Data Array for a summary of the array data.

The array has a maximum size of 5 elements.

### 3.38.2 Usage

During real-time execution, this array is not recomputed. TOW\_BURN\_SPEED\_DEG determines the number of elements of the array to be used in the polynomial evaluation.

### 3.38.2.1 Algorithm

Tow\_burn\_speed\_coeff is used to compute the ATGM missile speed at launch using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_atgm\_fire. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/

* Set the initial time, location, orientation, and speed of the generic

* missile.

/*/

mptr->time = 0.0;

vec_copy (launch_point, mptr->location);

mat_copy (loc_sight_to_world, mptr->orientation);

mptr->speed = missile_util_eval_poly (TOW_BURN_SPEED_DEG,

tow_burn_speed_coeff, 0.0) + launch_speed;

mptr->init_speed = launch_speed;
```

Tow\_burn\_speed\_coeff is used to compute the ATGM missile speed during powered flight [burn] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_atgm\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/

* Find the current missile speed and the cosines of the maximum allowed

* turn angles in each direction. The equations used are different before and

* after motor burnout.

/*/

if (time < TOW_BURNOUT_TIME)

{

mptr->speed = missile_util_eval_poly (TOW_BURN_SPEED_DEG,
tow_burn_speed_coeff, time) + mptr->init_speed;
missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
}

, else

{

mptr->speed = missile_util_eval_poly (TOW_COAST_SPEED_DEG,
tow_coast_speed_coeff, time) + mptr->init_speed;
missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
}
```

See APPENDIX F for a complete source code listing.

# 3.39 Tow\_coast\_speed\_coeff [for ATGM]

The tow\_coast\_speed\_coeff array consists of the coefficients for a polynomial equation defining the ATGM missile coast speed with respect to time in the form using the Newton-Raphson method. The tow missile source code was used as the baseline for the ATGM missile function; many of the ATGM constants, variables, CSCs and CSUs have the same name as in the TOW missile source code.

#### 3.39.1 Initialization

The tow\_coast\_speed\_coeff array is initialized during execution of the CSU missile\_atgm\_init, called by CSC weapons\_init. Execution of the CSU missile\_atgm\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.38. - ATGM Missile Coast Speed Coefficient Data Array for a summary of the array data.

The array has a maximum size of 5 elements.

### 3'.39.2 Usage

During real-time execution, this array is not recomputed. TOW\_COAST\_SPEED\_DEG determines the number of elements of the array to be used in the polynomial evaluation.

# 3.39.2.1 Algorithm

Tow\_coast\_speed\_coeff is used to compute the ATGM missile speed during unpowered flight [coast] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_atgm\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*/

* Find the current missile speed and the cosines of the maximum allowed

* turn angles in each direction. The equations used are different before and

* after motor burnout.

/*/

if (time < TOW_BURNOUT_TIME)

{

mptr->speed = missile_util_eval_poly (TOW_BURN_SPEED_DEG,
tow_burn_speed_coeff, time) + mptr->init_speed;
missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
}
else
```

# 3.40 Tow\_burn\_turn\_coeff [for ATGM]

The tow\_burn\_turn\_coeff two-dimensional array consists of the coefficients for three polynomial equations [sidewards, upwards, and downwards movement] defining the ATGM missile maximum cosine of turn while powered with respect to time in the form using the Newton-Raphson method. The tow missile source code was used as the baseline for the ATGM missile function; many of the ATGM constants, variables, CSCs and CSUs have the same name as in the TOW missile source code. A turn factor is used to scale the TOW coefficients for ATGM performance.

#### 3.40.1 Initialization

The tow\_burn\_turn\_coeff array is initialized during execution of the CSU missile\_atgm\_init, called by CSC weapons\_init. Execution of the CSU missile\_atgm\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.40. - ATGM Missile Burn Turn Coefficients Data Array for a summary of the array data.

```
/* Upwards turn. */
0.999960667258, /* a_0 - cos(rad)/tick */
-3.1492328e-6 /* a_1 - cos(rad)/tick**2 */
},
{
    /* Downwards turn. */
0.999978909989, /* a_0 - cos(rad)/tick */
-7.8194991e-9 /* a_1 - cos(rad)/tick**2 */
}
};
```

The array has a maximum size of 3 by 2 elements.

Changing this constant requires a recompile because of the hard coded multidimension characteristic.

### 3.40.2 Usage

During real-time execution, this array is not recomputed. The size of the array in the type definition for MAX\_COS\_COEFF determines the number of elements of the array to be used in the polynomial evaluation.

# 3.40.2.1 Algorithm

The tow\_burn\_turn\_coeff array is initialized and scaled for ATGM missile performance by a call to the CSU missile\_atgm\_init.

```
for (i=0; i<tow_coast_turn_coeff.deg; i++)
     {
      tow_coast_turn_coeff.side_coeff[i] *= ATGM_TURN_FACTOR;
      tow_coast_turn_coeff.up_coeff[i] *= ATGM_TURN_FACTOR;
      tow_coast_turn_coeff.down_coeff[i] *= ATGM_TURN_FACTOR;
    }</pre>
```

Tow\_burn\_turn\_coeff is used to compute the cosine of the maximum allowed turn angle in each axis for the ATGM missile during powered flight [burn] using the CSU missile\_util\_cos\_coeff, and called by the CSU missile\_atgm\_fly. The CSU missile\_util\_cos\_coeff uses the Newton-Raphson method to evaluate the polynomial with inputs of missile pointer, coefficient array, and time.

```
/*/

* Find the current missile speed and the cosines of the maximum allowed

* turn angles in each direction. The equations used are different before and

* after motor burnout.

/*/

if (time < TOW_BURNOUT_TIME)

{

mptr->speed = missile_util_eval_poly (TOW_BURN_SPEED_DEG,
tow_burn_speed_coeff, time) + mptr->init_speed;
missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
}

else

{

mptr->speed = missile_util_eval_poly (TOW_COAST_SPEED_DEG,
tow_coast_speed_coeff, time) + mptr->init_speed;
missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
}
```

See APPENDIX F for a complete source code listing.

# 3.41 Tow\_coast \_turn\_coeff [for ATGM]

The tow\_coast\_turn\_coeff two-dimensional array consists of the coefficients for three polynomial equations [sidewards, upwards, and downwards movement] defining the ATGM missile maximum cosine of turn while unpowered with respect to time in the form using the Newton-Raphson method. The tow missile source code was used as the baseline for the ATGM missile function; many of the ATGM constants, variables, CSCs and CSUs

have the same name as in the TOW missile source code. A turn factor is used to scale the TOW coefficients for ATGM performance.

#### 3.41.1 Initialization

The tow\_coast\_turn\_coeff array is initialized during execution of the CSU missile\_atgm\_init, called by CSC weapons\_init. Execution of the CSU missile\_atgm\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.41. - ATGM Missile Coast Turn Coefficients Data Array for a summary of the array data.

The array has a maximum size of 3 by 4 elements.

```
/*/
* Coefficients for the cosine of max turn polynomials after motor burnout.
/*/
static MAX_COS_COEFF tow_coast_turn_coeff =
               /* Order of the polynomials. */
  3,
              /* Sidewards turn. */
    0.99995112518, /* a_0 - cos(rad)/tick */
    8.96333e-7, /* a 1 - cos(rad)/tick**2 */
    -5.995375e-9, /* a_2 - cos(rad)/tick**3 */
    1.162225e-11 /* a_3 - cos(rad)/tick**4 */
  },
              /* Upwards turn. */
    0.9998498495, /* a 0 - cos(rad)/tick */
    1.657779e-6, /* a_1 - cos(rad)/tick**2 */
    -8.231861e-9, /* a_2 - cos(rad)/tick**3 */
    1.381832e-11 /* a 3 - cos(rad)/tick**4 */
  },
  {
              /* Downwards turn. */
    0.9999714014, /* a_0 - cos(rad)/tick */
    3.382077e-7, /* a 1 - cos(rad)/tick**2 */
    -1.601259e-9, /* a_2 - cos(rad)/tick**3 */
    2.623014e-12 /* a_3 - cos(rad)/tick**4 */
};
```

### 3.41.2 Usage

During real-time execution, this array is not recomputed. MAVERICK\_BURN\_SPEED\_DEG determines the number of elements of the array to be used in the polynomial evaluation.

# 3.41.2.1 Algorithm

The tow\_coast\_turn\_coeff array is initialized and scaled for ATGM missile performance by a call to the CSU missile\_atgm\_init.

Tow\_coast\_turn\_coeff is used to compute the cosine of the maximum allowed turn angle in each axis for the ATGM missile during unpowered flight [coast] using the CSU missile\_util\_cos\_coeff, and called by the CSU missile\_atgm\_fly. The CSU missile\_util\_cos\_coeff uses the Newton-Raphson method to evaluate the polynomial with inputs of missile pointer, coefficient array, and time.

```
/*/

* Find the current missile speed and the cosines of the maximum allowed

* turn angles in each direction. The equations used are different before and

* after motor burnout.

/*/

if (time < TOW_BURNOUT_TIME)

{
```

# 3.42 Kem\_miss\_char

The kem\_miss\_char array consists of characteristics and parameters describing a KEM missile system and its performance constraints. The KEM missile source code was derived from the ADAT missile source code.

# 3.42.1 KEM\_BURNOUT\_TIME

KEM\_BURNOUT\_TIME is a constant defining the time of powered flight for kem missile in ticks.

### 3.42.1 Initialization

The constant is initialized during execution of the CSU missile\_kem\_init, called by CSC weapons\_init. Execution of the CSU missile\_kem\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.42. - KEM Missile Characteristics Data Array for a summary of the constants data.

```
#define KEM_BURNOUT_TIME kem_miss_char[0]
```

### 3.42.2 Usage

During real-time execution, this constant is not recomputed.

# 3.42.2.1 Algorithm

KEM\_BURNOUT\_TIME is used to control computation of the missile flyout speed by a call to the CSC missile\_kem\_fly.

```
* Find the current missile speed and the cosines of the maximum allowed
* turn angles in each direction. The equations used are different before
* and after motor burnout.
*/
  if (time < KEM_BURNOUT_TIME)
     mptr->speed = (missile_util_eval_poly (KEM_BURN_SPEED_DEG,
          kem_burn_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
          mptr->init_speed;
     mptr->cos_max_turn[0] = missile_util_eval_poly (
          KEM_BURN_TURN_DEG, kem_burn_turn_coeff, time);
 }
  else
     mptr->speed = (missile_util_eval_poly (KEM_COAST_SPEED_DEG,
          kem_coast_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
          mptr->init_speed;
     mptr->cos_max_turn[0] = missile_util_eval_poly (
          KEM_COAST_TURN_DEG, kem_coast_turn_coeff, time);
  }
```

# 3.42.2 KEM\_MAX\_FLIGHT\_TIME

KEM\_MAX\_FLIGHT\_TIME is a constant defining the maximum flight time for the KEM missile in ticks.

### 3.42.1 Initialization

The constant is initialized during execution of the CSU missile\_kem\_init, called by CSC weapons\_init. Execution of the CSU missile\_kem\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.42. - KEM Missile Characteristics Data Array for a summary of the constants data.

```
#define KEM_MAX_FLIGHT_TIME kem_miss_char[1]
```

# 3.42.2 Usage

During real-time execution, this constant is not recomputed.

### 3.42.2.1 Algorithm

KEM\_MAX\_FLIGHT\_TIME is used to initialize the maximum flight time for an individual KEM missile by a call to the CSU missile\_kem\_init.

```
for (i = 0; i < num_missiles; i++)
{
    kem_array[i].mptr.state = KEM_FREE;
    kem_array[i].mptr.max_flight_time = KEM_MAX_FLIGHT_TIME;
    kem_array[i].mptr.max_turn_directions = 1;
}</pre>
```

See APPENDIX H for a complete source code listing.

### 3.42.3 KEM\_TO\_MACH5\_FACTOR

KEM\_TO\_MACH5\_FACTOR is a constant defining the speed factor to raise missile performance from ADAT to KEM; just after burnout, the ADAT has a maximum velocity of 230 m/sec, while the KEM has a maximum velocity of 1524 m/sec.

#### 3.42.1 Initialization

The constant is initialized during execution of the CSU missile\_kem\_init, called by CSC weapons\_init. Execution of the CSU missile\_kem\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.42. - KEM Missile Characteristics Data Array for a summary of the constants data.

```
#define KEM_TO_MACH5_FACTOR kem_miss_char[2]
```

### 3.42.2 Usage

During real-time execution, this constant is not recomputed.

# 3.42.2.1 Algorithm

KEM\_TO\_MACH5\_FACTOR is used to scale the burn speed coefficients when the launch speed is computed by a call to the CSU missile\_kem\_fire.

```
/*

* Set the initial time, location, orientation, and speed of the generic

* missile.

*/

mptr->time = 0.0;

vec_copy (launch_point, mptr->location);

mat_copy (loc_sight_to_world, mptr->orientation);

mptr->speed = (missile_util_eval_poly (KEM_BURN_SPEED_DEG,

kem_burn_speed_coeff, 0.0) * KEM_TO_MACH5_FACTOR) +

launch_speed;

mptr->init_speed = launch_speed;

if (kptr->target_vehicle_id.vehicle == vehicleIrrelevant)

comm_target_type = targetIsVehicle;

comm_target_type = targetIsVehicle;
```

KEM\_TO\_MACH5\_FACTOR is used to scale the burn speed and coast speed coefficients when the missile flyout speed is computed by a call to the CSU missile\_kem\_fly.

# 3.43 Kem\_miss\_poly\_deg

The kem\_miss\_poly\_deg array consists of values of the degree of each polynomial equation used to compute the burn speed, the coast speed, maximum cosines of turns while powered, and maximum cosines of turns while unpowered for the KEM missile. The KEM missile source code was derived from the ADAT missile source code.

# 3.43.1 KEM\_BURN\_SPEED\_DEG

KEM\_BURN\_SPEED\_DEG is a constant defining the polynomial degree for the KEM missile burn speed coefficient data array. KEM\_BURN\_SPEED\_DEG is the first element of the kem\_miss\_poly\_deg array.

#### 3.43.1.1 Initialization

The constant is initialized during execution of the CSU missile\_kem\_init, called by CSC weapons\_init. Execution of the CSU missile\_kem\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.43. - KEM Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define KEM_BURN_SPEED_DEG kem_miss_poly_deg[0]
```

# 3.43.1.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for KEM\_BURN\_SPEED\_DEG is 9, especially, the declared size of the kem\_burn\_speed\_coeff array is 10.

# 3.43.1.2.1 Algorithm

KEM\_BURN\_SPEED\_DEG is used to compute the KEM missile speed at launch using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_kem\_fire. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*

* Set the initial time, location, orientation, and speed of the generic

* missile.

*/

mptr->time = 0.0;

vec_copy (launch_point, mptr->location);

mat_copy (loc_sight_to_world, mptr->orientation);

mptr->speed = (missile_util_eval_poly (KEM_BURN_SPEED_DEG,

kem_burn_speed_coeff, 0.0) * KEM_TO_MACH5_FACTOR) +

launch_speed;

mptr->init_speed = launch_speed;

if (kptr->target_vehicle_id.vehicle == vehicleIrrelevant)

comm_target_type = targetUnknown;

else

comm_target_type = targetIsVehicle;
```

KEM\_BURN\_SPEED\_DEG is used to compute the KEM missile speed during powered flight [burn] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_kem\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*

* Find the current missile speed and the cosines of the maximum allowed

* turn angles in each direction. The equations used are different before

* and after motor burnout.

*/

if (time < KEM_BURNOUT_TIME)

{

mptr->speed = (missile_util_eval_poly (KEM_BURN_SPEED_DEG, kem_burn_speed_coeff, time) * KEM_TO_MACH5_FACTOR) + mptr->init_speed;
```

# 3.43.2 KEM\_COAST\_SPEED\_DEG

KEM\_COAST\_SPEED\_DEG is a constant defining the polynomial degree for the kem missile coast speed coefficient data array. KEM\_COAST\_SPEED\_DEG is the second element of the kem\_miss\_poly\_deg array.

### 3.43.2.1 Initialization

The constant is initialized during execution of the CSU missile\_kem\_init, called by CSC weapons\_init. Execution of the CSU missile\_kem\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.43. - KEM Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define KEM_COAST_SPEED_DEG kem_miss_poly_deg[1]
```

# 3.43.2.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for KEM\_COAST\_SPEED\_DEG is 9, especially, the declared size of the kem\_burn\_speed\_coeff array is 10.

# 3.43.2.2.1 Algorithm

KEM\_COAST\_SPEED\_DEG is used to compute the KEM missile speed during unpowered flight [coast] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_kem\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
* Find the current missile speed and the cosines of the maximum allowed
* turn angles in each direction. The equations used are different before
* and after motor burnout.
  if (time < KEM_BURNOUT_TIME)
     mptr->speed = (missile_util_eval_poly (KEM_BURN_SPEED_DEG,
          kem_burn_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
          mptr->init_speed;
     mptr->cos_max_turn[0] = missile_util_eval_poly (
          KEM_BURN_TURN_DEG, kem_burn_turn_coeff, time);
 }
 else
     mptr->speed = (missile_util_eval_poly (KEM_COAST_SPEED_DEG,
          kem_coast_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
          mptr->init_speed;
     mptr->cos_max_turn[0] = missile_util_eval_poly (
          KEM_COAST_TURN_DEG, kem_coast_turn_coeff, time);
```

# 3.43.3 KEM\_BURN\_TURN\_DEG

KEM\_BURN\_TURN\_DEG is a constant defining the polynomial degree for the cosine of the KEM missile maximum allowed turn angle, burn turn coefficient data array. KEM\_BURN\_TURN\_DEG is the third element of the kem\_miss\_poly\_deg array.

### 3.43.3.1 Initialization

The constant is initialized during execution of the CSU missile\_kem\_init, called by CSC weapons\_init. Execution of the CSU missile\_kem\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.43. - KEM Missile Polynomial Degree Data Array for a summary of the constants data.

#define KEM\_BURN\_TURN\_DEG kem\_miss\_poly\_deg[2]

#### 3.43.3.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for KEM\_BURN\_TURN\_DEG is 9, especially, the declared size of the kem\_burn\_speed\_coeff array is 10.

# 3.43.3.2.1 Algorithm

KEM\_BURN\_TURN\_DEG is used to compute the cosine of the maximum allowed turn angle for the KEM missile during powered flight [burn] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_kem\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
* Find the current missile speed and the cosines of the maximum allowed
* turn angles in each direction. The equations used are different before
* and after motor burnout.
  if (time < KEM_BURNOUT_TIME)
     mptr->speed = (missile_util_eval_poly (KEM_BURN_SPEED_DEG,
          kem_burn_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
          mptr->init_speed;
     mptr->cos_max_turn[0] = missile_util_eval_poly (
          KEM_BURN_TURN_DEG, kem_burn_turn_coeff, time);
 }
 else
     mptr->speed = (missile_util_eval_poly (KEM_COAST_SPEED_DEG,
          kem_coast_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
          mptr->init_speed;
     mptr->cos_max_turn[0] = missile_util_eval_poly (
          KEM_COAST_TURN_DEG, kem_coast_turn_coeff, time);
 }
```

See APPENDIX H for a complete source code listing.

# 3.43.4 KEM\_COAST\_TURN\_DEG

KEM\_COAST\_TURN\_DEG is a constant defining the polynomial degree for the cosine of the KEM missile maximum allowed turn angle, coast turn coefficient data array. KEM\_COAST\_TURN\_DEG is the fourth element of the kem\_miss\_poly\_deg array.

#### 3.43.4.1 Initialization

The constant is initialized during execution of the CSU missile\_kem\_init, called by CSC weapons\_init. Execution of the CSU missile\_kem\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.43. - KEM Missile Polynomial Degree Data Array for a summary of the constants data.

```
#define KEM_COAST_TURN_DEG kem_miss_poly_deg[3]
```

### 3.43.4.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for KEM\_COAST\_TURN\_DEG is 9, especially, the declared size of the kem\_burn\_speed\_coeff array is 10.

### 3.43.4.2.1 Algorithm

KEM\_COAST\_TURN\_DEG is used to compute the cosine of the maximum allowed turn angle for the KEM missile during unpowered flight [coast] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_kem\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

### 3.44 Kem\_burn\_speed\_coeff -

The kem\_burn\_speed\_coeff array consists of the coefficients for a polynomial equation defining the KEM missile burn speed with respect to time in the form using the Newton-Raphson method. The KEM missile source code was derived from the ADAT missile source code.

#### 3.44.1 Initialization

The kem\_burn\_speed\_coeff array is initialized during execution of the CSU missile\_kem\_init, called by CSC weapons\_init. Execution of the CSU missile\_kem\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.44. - KEM Missile Burn Speed Coefficient Data Array for a summary of the array data.

```
/*/

* Coefficients for the speed polynomial before motor burnout initialized

* to default values.

/*/
```

The array has a maximum size of 10 elements.

### 3.44.2 Usage

During real-time execution, this array is not recomputed. KEM\_BURN\_SPEED\_DEG determines the number of elements of the array to be used in the polynomial evaluation.

# 3.44.2.1 Algorithm

The kem\_burn\_speed\_coeff array is used to compute the initial speed at launch of the KEM missile using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_kem\_fire. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

The kem\_burn\_speed\_coeff array is used to compute the KEM missile speed during powered flight [burn] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_kem\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
* Find the current missile speed and the cosines of the maximum allowed
* turn angles in each direction. The equations used are different before
* and after motor burnout.
*/
  if (time < KEM_BURNOUT_TIME)
     mptr->speed = (missile_util_eval_poly (KEM_BURN_SPEED_DEG,
          kem_burn_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
          mptr->init_speed;
     mptr->cos_max_turn[0] = missile_util_eval_poly (
          KEM_BURN_TURN_DEG, kem_burn_turn_coeff, time);
 }
  else
     mptr->speed = (missile_util_eval_poly (KEM_COAST_SPEED_DEG,
          kem_coast_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
          mptr->init_speed;
     mptr->cos_max_turn[0] = missile_util_eval_poly (
          KEM_COAST_TURN_DEG, kem_coast_turn_coeff, time);
```

# 3.45 Kem\_coast\_speed\_coeff

The kem\_coast\_speed\_coeff array consists of the coefficients for a polynomial equation defining the KEM missile coast speed with respect to time in the form using the Newton-Raphson method. The KEM missile source code was derived from the ADAT missile source code.

#### 3.45.1 Initialization

The kem\_coast\_speed\_coeff array is initialized during execution of the CSU missile\_kem\_init, called by CSC weapons\_init. Execution of the CSU missile\_kem\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.45. - KEM Missile Coast Speed Coefficient Data Array for a summary of the array data.

The array has a maximum size of 10 elements.

### 3.45.2 Usage

During real-time execution, this array is not recomputed. KEM\_COAST\_SPEED\_DEG determines the number of elements of the array to be used in the polynomial evaluation.

# 3.45.2.1 Algorithm

The kem\_coast\_speed\_coeff array is used to compute the KEM missile speed during unpowered flight [coast] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_kem\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
/*

* Find the current missile speed and the cosines of the maximum allowed

* turn angles in each direction. The equations used are different before

* and after motor burnout.

*/

if (time < KEM_BURNOUT_TIME)

{

mptr->speed = (missile_util_eval_poly (KEM_BURN_SPEED_DEG, kem_burn_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
```

### 3.46 Kem\_burn\_turn\_coeff

The kem\_burn\_turn\_coeff array consists of the coefficients for a polynomial equation defining the KEM missile maximum cosine of the turn angle while in powered flight with respect to time in the form using the Newton-Raphson method. The KEM missile source code was derived from the ADAT missile source code.

#### 3.46.1 Initialization

The kem\_burn\_turn\_coeff array is initialized during execution of the CSU missile\_kem\_init, called by CSC weapons\_init. Execution of the CSU missile\_kem\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.46. - KEM Missile Burn Turn Coefficient Data Array for a summary of the array data.

```
/*/
* Coefficients for the cosine of max turn polynomial before motor burnout.
/*/
static REAL kem_burn_turn_coeff[10] =
{
0.999993,     /* a_0 - cos(rad)/tick */
-6.2386917e-7,     /* a_1 - cos(rad)/tick**2 */
1.6146426e-7,     /* a_2 - cos(rad)/tick**3 */
-9.720142e-7,     /* a_3 - cos(rad)/tick**4 */
0.0,
```

```
0.0,
0.0,
0.0,
0.0,
0.0
```

The array has a maximum size of 10 elements.

### 3.46.2 Usage

During real-time execution, this array is not recomputed. KEM\_BURN\_TURN\_DEG determines the number of elements of the array to be used in the polynomial evaluation.

### 3.46.2.1 Algorithm

The kem\_burn\_turn\_coeff array is used to compute the cosine of the maximum allowed turn angle for the KEM missile during powered flight [burn] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_kem\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

# 3.47 Kem\_coast\_turn\_coeff

The kem\_coast\_turn\_coeff array consists of the coefficients for a polynomial equation defining the KEM missile maximum cosine of the turn angle while in unpowered flight with respect to time in the form using the Newton-Raphson method. The KEM missile source code was derived from the ADAT missile source code.

### 3.47.1 Initialization

The kem\_coast\_turn\_coeff array is initialized during execution of the CSU missile\_kem\_init, called by CSC weapons\_init. Execution of the CSU missile\_kem\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.47. - KEM Missile Coast Turn Coefficient Data Array for a summary of the array data.

```
0.0,
0.0,
0.0,
0.0
};
```

The array has a maximum size of 10 elements.

### 3.47.2 Usage

During real-time execution, this array is not recomputed. MAVERICK\_BURN\_SPEED\_DEG determines the number of elements of the array to be used in the polynomial evaluation.

# 3.47.2.1 Algorithm

The kem\_coast\_turn\_coeff array is used to compute the cosine of the maximum allowed turn angle for the KEM missile during unpowered flight [coast] using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_kem\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and time.

```
* Find the current missile speed and the cosines of the maximum allowed
* turn angles in each direction. The equations used are different before
* and after motor burnout.
*/
  if (time < KEM_BURNOUT_TIME)
     mptr->speed = (missile_util_eval_poly (KEM_BURN_SPEED_DEG,
         kem_burn_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
          mptr->init_speed;
     mptr->cos_max_turn[0] = missile_util_eval_poly (
          KEM_BURN_TURN_DEG, kem_burn_turn_coeff, time);
 }
 else
 {
     mptr->speed = (missile_util_eval_poly (KEM_COAST_SPEED_DEG,
          kem_coast_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
          mptr->init_speed;
     mptr->cos_max_turn[0] = missile_util_eval_poly (
          KEM_COAST_TURN_DEG, kem_coast_turn_coeff, time);
```

# 3.48 Nlos\_miss\_char

The nlos\_miss\_char array consists of characteristics and parameters describing an NLOS missile system and its performance constraints.

# 3.48.1 NLOS\_LOCK\_THRESHOLD

NLOS\_LOCK\_THRESHOLD is a constant defining the threshold lock for the NLOS missile

#### 3.48.1.1 Initialization

The constant is initialized during execution of the CSU missile\_nlos\_init, called by CSC weapons\_init. Execution of the CSU missile\_nlos\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

#define NLOS\_LOCK\_THRESHOLD

nlos\_miss\_char[ 0]

### 3.48.1.2 Usage

During real-time execution, this constant is not recomputed.

# 3.48.1.2.1 Algorithm

NLOS\_LOCK\_THRESHOLD is used to compute the vector to the preferred vehicle by a call to the CSU near\_get\_preferred\_veh\_near\_vector in a call to the CSC missile\_nlos\_fly.

```
/*/

* choose the correct targeting option depending on flight time

/*/

if (time == NLOS_LEVEL_FLIGHT_TIME)

printf("extra_waypoint: %f %f %f\n",

mptr->location[0],

mptr->location[1],

mptr->location[2]);

if (time < NLOS_VERTICAL_FLIGHT_TIME)
```

```
missile_nlos_fly_to_point(mptr, peak_target);
else if (time < NLOS_DECLINE_FLIGHT_TIME)
   missile_nlos_fly_to_point(mptr, decline_target);
else if (time < NLOS_LEVEL_FLIGHT_TIME)
   level_target[Z] = mptr->location[Z];
   missile_nlos_fly_to_point(mptr, level_target);
else
   switch (target_scheme)
       case NLOS_FLY_TO_POINT_IN_SPACE:
          missile_nlos_fly_to_point(mptr, nlos_target_loc);
          break;
       case NLOS_FLY_TO_POINT_RELATIVE:
          missile_target_nlos(mptr, nlos_target_loc);
          break;
       case NLOS_FLY_TO_TARGET:
          target = near_get_preferred_veh_near_vector (
                            &nlos_target_id,
                            RVA ALL_VEH,
                            mptr->location,
                            mptr->orientation[1],
                            NLOS_LOCK_THRESHOLD,
                            &nlos_req_id);
          if (target != NULL)
             timed_printf("miss_nlos: target locked on\n");
       missile target_pursuit (mptr, target);
          else
             missile_target_unguided(mptr);
          break;
      default:
          printf("missile_nlos_fly: bad target_scheme\n");
          break;
      }
    }
```

# 3.48.2 NLOS\_MAX\_TURN\_ANGLE

NLOS\_MAX\_TURN\_ANGLE is a constant defining the maximum turn angle for the NLOS missile.

## 3.48.2.1 Initialization

The constant is initialized during execution of the CSU missile\_nlos\_init, called by CSC weapons\_init. Execution of the CSU missile\_nlos\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

#define NLOS\_MAX\_TURN\_ANGLE

nlos\_miss\_char[ 1]

### 3.48.2.2 Usage

During real-time execution, this constant is not recomputed.

# 3.48.2.2.1 Algorithm

NLOS\_MAX\_TURN\_ANGLE is used to compute the cosine of the maximum turn angle for the NLOS missile by a call to the CSU missile\_nlos\_init.

mptr->cos\_max\_turn[0] = cos (NLOS\_MAX\_TURN\_ANGLE);

See APPENDIX J for a complete source code listing.

# 3.48.3 NLOS\_VERTICAL\_FLIGHT\_TIME

NLOS\_VERTICAL\_FLIGHT\_TIME is a constant defining the flight time in the vertical mode for the NLOS missile.

## 3.48.3.1 Initialization

The constant is initialized during execution of the CSU missile\_nlos\_init, called by CSC weapons\_init. Execution of the CSU missile\_nlos\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

#define NLOS\_VERTICAL\_FLIGHT\_TIME nlos\_miss\_char[ 2]

### 3.48.3.2 Usage

During real-time execution, this constant is not recomputed.

## 3.48.3.2.1 Algorithm

NLOS\_VERTICAL\_FLIGHT\_TIME is used control the flight path of the NLOS missile by a call to the CSC missile\_nlos\_fly.

```
/*/
* choose the correct targeting option depending on flight time
if (time == NLOS_LEVEL_FLIGHT_TIME)
, printf("extra_waypoint: %f %f %f \n",
     mptr->location[0],
     mptr->location[1],
     mptr->location[2]);
  if (time < NLOS_VERTICAL_FLIGHT_TIME)
      missile_nlos_fly_to_point(mptr, peak_target);
  else if (time < NLOS_DECLINE_FLIGHT_TIME)
      missile_nlos_fly_to_point(mptr, decline_target);
  else if (time < NLOS_LEVEL_FLIGHT_TIME)
      level target[Z] = mptr->location[Z];
      missile_nlos_fly_to_point(mptr, level_target);
  else
      switch (target_scheme)
         case NLOS_FLY_TO_POINT_IN_SPACE:
            missile_nlos_fly_to_point(mptr, nlos_target_loc);
            break;
         case NLOS_FLY_TO_POINT_RELATIVE:
            missile_target_nlos(mptr, nlos_target_loc);
            break;
```

```
case NLOS FLY TO TARGET:
   target = near_get_preferred_veh_near_vector (
                      &nlos_target_id,
                      RVA_ALL_VEH,
                      mptr->location,
                      mptr->orientation[1],
                     NLOS_LOCK_THRESHOLD,
                      &nlos_req_id);
   if (target != NULL)
       timed_printf("miss_nlos: target locked on\n");
missile_target_pursuit (mptr, target);
   else
      missile_target_unguided(mptr);
   break;
default:
   printf("missile_nlos_fly: bad target_scheme\n");
   break;
```

# 3.48.4 NLOS\_DECLINE\_FLIGHT\_TIME

NLOS\_DECLINE\_FLIGHT\_TIME is a constant defining the flight time in the decline mode for the NLOS missile.

#### 3.48.4.1 Initialization

The constant is initialized during execution of the CSU missile\_nlos\_init, called by CSC weapons\_init. Execution of the CSU missile\_nlos\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

```
#define NLOS_DECLINE_FLIGHT_TIME nlos_miss_char[ 3]
```

#### 3.48.4.2 Usage

During real-time execution, this constant is not recomputed.

# 3.48.4.2.1 Algorithm

NLOS\_DECLINE\_FLIGHT\_TIME is used control the flight path of the NLOS missile by a call to the CSC missile\_nlos\_fly.

```
* choose the correct targeting option depending on flight time
/*/
if (time == NLOS_LEVEL_FLIGHT_TIME)
  printf("extra_waypoint: %f %f %f\n",
      mptr->location[0],
      mptr->location[1],
      mptr->location[2]);
   if (time < NLOS_VERTICAL_FLIGHT_TIME)
      missile_nlos_fly_to_point(mptr, peak_target);
  else if (time < NLOS_DECLINE_FLIGHT_TIME)
      missile_nlos_fly_to_point(mptr, decline_target);
   else if (time < NLOS_LEVEL_FLIGHT_TIME)
      level target[Z] = mptr->location[Z];
      missile_nlos_fly_to_point(mptr, level_target);
  else
      switch (target_scheme)
         case NLOS_FLY_TO_POINT_IN_SPACE:
            missile_nlos_fly_to_point(mptr, nlos_target_loc);
            break;
         case NLOS_FLY_TO_POINT_RELATIVE:
            missile_target_nlos(mptr, nlos_target_loc);
            break;
```

```
case NLOS_FLY_TO_TARGET:
   target = near_get_preferred_veh_near_vector (
                      &nlos_target_id,
                      RVA_ALL_VEH,
                      mptr->location,
                      mptr->orientation[1],
                      NLOS_LOCK_THRESHOLD,
                      &nlos_req_id);
   if (target != NULL)
       timed_printf("miss_nlos: target locked on\n");
missile_target_pursuit (mptr, target);
   else
      missile_target_unguided(mptr);
   break;
default:
   printf("missile_nlos_fly: bad target_scheme\n");
   break;
```

# 3.48.5 NLOS\_LEVEL\_FLIGHT\_TIME

NLOS\_LEVEL\_FLIGHT\_TIME is a constant defining the flight time in the level mode for the NLOS missile.

#### 3.48.5.1 Initialization

The constant is initialized during execution of the CSU missile\_nlos\_init, called by CSC weapons\_init. Execution of the CSU missile\_nlos\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

#define NLOS\_LEVEL\_FLIGHT\_TIME

nlos\_miss\_char[ 4]

### 3.48.5.2 Usage

During real-time execution, this constant is not recomputed.

## 3.48.5.2.1 Algorithm

NLOS\_LEVEL\_FLIGHT\_TIME is used control the flight path of the NLOS missile by a call to the CSC missile\_nlos\_fly.

```
/*/
* choose the correct targeting option depending on flight time
if (time == NLOS_LEVEL_FLIGHT_TIME)
  printf("extra_waypoint: %f %f %f \n",
      mptr->location[0],
      mptr->location[1],
     mptr->location[2]);
  if (time < NLOS_VERTICAL_FLIGHT_TIME)
     missile_nlos_fly_to_point(mptr, peak_target);
  else if (time < NLOS_DECLINE_FLIGHT_TIME)
      missile_nlos_fly_to_point(mptr, decline_target);
  else if (time < NLOS_LEVEL_FLIGHT_TIME)
      level_target[Z] = mptr->location[Z];
      missile_nlos_fly_to_point(mptr, level_target);
  else
      switch (target_scheme)
         case NLOS FLY TO POINT_IN_SPACE:
            missile_nlos_fly_to_point(mptr, nlos_target_loc);
            break;
         case NLOS_FLY_TO_POINT_RELATIVE:
            missile_target_nlos(mptr, nlos_target_loc);
            break;
```

```
case NLOS_FLY_TO_TARGET:
     target = near_get_preferred_veh_near_vector (
                        &nlos_target_id,
                        RVA_ALL_VEH,
                        mptr->location,
                        mptr->orientation[1],
                        NLOS_LOCK_THRESHOLD,
                        &nlos_req_id);
     if (target != NULL)
        timed_printf("miss_nlos: target locked on\n");
  missile_target_pursuit (mptr, target);
     else
        missile_target_unguided(mptr);
     break;
  default:
      printf("missile_nlos_fly: bad target_scheme\n");
      break;
 }
}
```

# 3.48.6 NLOS\_ARM\_TIME

NLOS\_ARM\_TIME is a constant defining the nlos missile arm time delay before firing in ticks.

# 3.48.6.1 Initialization

The constant is initialized during execution of the CSU missile\_nlos\_init, called by CSC weapons\_init. Execution of the CSU missile\_nlos\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

#define NLOS\_ARM\_TIME

nlos\_miss\_char[5]

### 3.48.6.2 Usage

During real-time execution, this constant is not recomputed.

## 3.48.6.2.1 Algorithm

NLOS\_ARM\_TIME is not used in the current calculations.

See APPENDIX J for a complete source code listing.

### 3.48.7 NLOS\_BURNOUT\_TIME

NLOS\_BURNOUT\_TIME is a constant defining the time of powered flight for the nlos missile in ticks.

#### 3.48.7.1 Initialization

The constant is initialized during execution of the CSU missile\_nlos\_init, called by CSC weapons\_init. Execution of the CSU missile\_nlos\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

0.1.0	NIT OC	DIIDNICIT	TITE ATT
#aerine	MLOS	_BURNOUT_	TIME

nlos\_miss\_char[6]

## 3.48.7.2 Usage

During real-time execution, this constant is not recomputed.

# 3.48.7.2.1 Algorithm

NLOS\_BURNOUT\_TIME is not used in the current calculations.

See APPENDIX J for a complete source code listing.

# 3.48.8 NLOS\_MAX\_FLIGHT\_TIME

NLOS\_MAX\_FLIGHT\_TIME is a constant defining the maximum flight time for the nlos missile assumed in ticks.

#### 3.48.8.1 Initialization

The constant is initialized during execution of the CSU missile\_nlos\_init, called by CSC weapons\_init. Execution of the CSU missile\_nlos\_init is normally done only once during CSCI initialization and is performed

Reference # W003092; 31 March 1993 VOLUME 1 of 3; Rev. 0.0

sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

#define NLOS\_MAX\_FLIGHT\_TIME nlos\_miss\_char[ 7]

### 3.48.8.2 Usage

During real-time execution, this constant is not recomputed.

## 3.48.8.2.1 Algorithm

NLOS\_MAX\_FLIGHT\_TIME is used to initialize the maximum flight time for the NLOS missile in the CSU missile\_nlos\_init.

mptr->max\_flight\_time = NLOS\_MAX\_FLIGHT\_TIME;

See APPENDIX J for a complete source code listing.

## 3.48.9 SPEED\_0

SPEED\_0 is a constant defining the reference speed for the NLOS missile.

#### 3.48.9.1 Initialization

The constant is initialized during execution of the CSU missile\_nlos\_init, called by CSC weapons\_init. Execution of the CSU missile\_nlos\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

#define SPEED\_0 nlos\_miss\_char[ 8]

# 3.48.9.2 Usage

During real-time execution, this constant is not recomputed.

# 3.48.9.2.1 Algorithm

SPEED\_0 is used to initialize the speed for the NLOS missile in calls to the CSU missile\_nlos\_init and the CSU missile\_nlos\_fire.

```
mptr->speed = SPEED_0;
```

### 3.48.10 SPEED\_1

SPEED\_1 is a constant defining the second speed profile of the NLOS missile during flight.

#### 3.48.10.1 Initialization

The constant is initialized during execution of the CSU missile\_nlos\_init, called by CSC weapons\_init. Execution of the CSU missile\_nlos\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

```
#define SPEED_1 nlos_miss_char[ 9]
```

### 3.48.10.2 Usage

During real-time execution, this constant is not recomputed.

# 3.48.10.2.1 Algorithm

SPEED\_1 is used to initialize the NLOS flight speed during the second phase of the flyout after time from launch exceeds 800 ticks.

```
/*/

* Set and _time_. This is created mostly for increased readablity.

/*/

time = mptr->time;

if (time > 800.0)

mptr->speed = SPEED_1;
```

See APPENDIX J for a complete source code listing.

## 3.48.11 THETA\_0

THETA\_0 is a constant defining the reference maximum turn angle which is scaled for speed.

# 3.48.11.1 Initialization

The constant is initialized during execution of the CSU missile\_nlos\_init, called by CSC weapons\_init. Execution of the CSU missile\_nlos\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

1/ #Geine IIICIA O 0.0100 1=110 /	2/2/2/4
	962634*/
#define THETA_0 nlos_miss	_char[10]

## 3.48.11.2 Usage

During real-time execution, this constant is not recomputed.

# 3.48.11.2.1 Algorithm

THETA\_0 is not used in the current calculations.

See APPENDIX J for a complete source code listing.

# 3.48.12 SIN\_UNGUIDE

SIN\_UNGUIDE is a constant defining the sine of level flight [4.0 degrees pitch] for an unguided nlos missile.

## 3.48.12.1 Initialization

The constant is initialized during execution of the CSU missile\_nlos\_init, called by CSC weapons\_init. Execution of the CSU missile\_nlos\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

#define SIN\_UNGUIDE nlos\_miss\_char[11]

#### 3.48.12.2 Usage

During real-time execution, this constant is not recomputed.

## 3.48.12.2.1 Algorithm

SIN\_UNGUIDE is not used in the current calculations.

See APPENDIX J for a complete source code listing.

### 3.48.13 COS\_UNGUIDE

COS\_UNGUIDE is a constant defining the cosine of level flight [4.0 degrees pitch] for an unguided nlos missile.

#### 3.48.13.1 Initialization

The constant is initialized during execution of the CSU missile\_nlos\_init, called by CSC weapons\_init. Execution of the CSU missile\_nlos\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

#define COS\_UNGUIDE

nlos\_miss\_char[12]

### 3.48.13.2 Usage

During real-time execution, this constant is not recomputed.

## 3.48.13.2.1 Algorithm

COS\_UNGUIDE is not used in the current calculations.

See APPENDIX J for a complete source code listing.

## 3.48.14 SIN\_CLIMB

SIN\_CLIMB is a constant defining the sine of the delta pitch angle [3.5 degrees] for a climbing nlos missile.

#### 3.48.14.1 Initialization

The constant is initialized during execution of the CSU missile\_nlos\_init, called by CSC weapons\_init. Execution of the CSU missile\_nlos\_init is normally done only once during CSCI initialization and is performed

sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

#define SIN\_CLIMB nlos\_miss\_char[13]

## 3.48.14.2 Usage

During real-time execution, this constant is not recomputed.

# 3.48.14.2.1 Algorithm

SIN\_CLIMB is not used in the current calculations.

See APPENDIX J for a complete source code listing.

# 3.48.15 COS\_CLIMB

COS\_CLIMB is a constant defining the cosine of the delta pitch angle [3.5 \*\* degrees] for a climbing nlos missile.

## 3.48.15.1 Initialization

The constant is initialized during execution of the CSU missile\_nlos\_init, called by CSC weapons\_init. Execution of the CSU missile\_nlos\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

#define COS_CLIMB	nlos_miss_char[14]

# 3.48.15.2 Usage

During real-time execution, this constant is not recomputed.

# 3.48.15.2.1 Algorithm

COS\_CLIMB is not used in the current calculations.

See APPENDIX J for a complete source code listing.

# 3.48.16 SIN\_LOCK

SIN\_LOCK is a constant defining the sine of the lock cone angle [9.0 degrees] for a locked-on nlos missile.

## 3.48.16.1 Initialization

The constant is initialized during execution of the CSU missile\_nlos\_init, called by CSC weapons\_init. Execution of the CSU missile\_nlos\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

#define SIN\_LOCK

nlos\_miss\_char[15]

### 3.48.16.2 Usage

During real-time execution, this constant is not recomputed.

# 3.48.16.2.1 Algorithm

SIN\_LOCK is not used in the current calculations.

See APPENDIX J for a complete source code listing.

# 3.48.17 COS\_LOCK

COS\_LOCK is a constant defining the cosine of the lock cone angle [9.0 degrees] for a locked-on nlos missile.

### 3.48.17.1 Initialization

The constant is initialized during execution of the CSU missile\_nlos\_init, called by CSC weapons\_init. Execution of the CSU missile\_nlos\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

#define COS\_LOCK

nlos\_miss\_char[16]

# 3.48.17.2 Usage

During real-time execution, this constant is not recomputed.

# 3.48.17.2.1 Algorithm

COS\_LOCK is not used in the current calculations.

### 3.48.18 COS\_TERM

COS\_TERM is a constant defining the cosine of the terminal angle [0.0 degrees] for a locked-on nlos missile.

#### 3.48.18.1 Initialization

The constant is initialized during execution of the CSU missile\_nlos\_init, called by CSC weapons\_init. Execution of the CSU missile\_nlos\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

#define	COS	TERM

nlos\_miss\_char[17]

### 3.48.18.2 Usage

During real-time execution, this constant is not recomputed.

### 3.48.18.2.1 Algorithm

COS\_TERM is not used in the current calculations.

See APPENDIX J for a complete source code listing.

# 3.48.19 **COS\_LOSE**

COS\_LOSE is a constant defining the cosine of the angle [20.0 degrees] for a loss-of-lock-on nlos missile.

#### 3.48.19.1 Initialization

The constant is initialized during execution of the CSU missile\_nlos\_init, called by CSC weapons\_init. Execution of the CSU missile\_nlos\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.48. - NLOS Missile Characteristics Data Array for a summary of the constants data.

#define COS\_LOSE

nlos miss\_char[18]

### 3.48.19.2 Usage

During real-time execution, this constant is not recomputed.

## 3.48.19.2.1 Algorithm

COS\_LOSE is not used in the current calculations.

See APPENDIX J for a complete source code listing.

## 3.49 Nlos\_miss\_poly\_deg

The nlos\_miss\_poly\_deg array consists of values of the degree of each polynomial equation used to compute the burn speed, and the coast speed for the NLOS missile.

### 3.49.1 NLOS\_BURN\_SPEED\_DEG

NLOS\_BURN\_SPEED\_DEG is a constant defining the polynomial degree for the NLOS missile burn speed coefficient data array

#### 3.49.1.1 Initialization

The constant is initialized during execution of the CSU missile\_nlos\_init, called by CSC weapons\_init. Execution of the CSU missile\_nlos\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.49. - NLOS Missile Polynomial Degree Data Array for a summary of the constants data.

#define NLOS\_BURN\_SPEED\_DEG nlos\_miss\_poly\_deg[0]

### 3.49.1.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for NLOS\_BURN\_SPEED\_DEG is 4, especially, the declared size of the nlos burn speed\_coeff array is 5.

# 3.49.1.2.1 Algorithm

NLOS\_BURN\_SPEED\_DEG is not used in the current calculations.

See APPENDIX J for a complete source code listing.

# 3.49.2 NLOS\_COAST\_SPEED\_DEG

NLOS\_COAST\_SPEED\_DEG is a constant defining the polynomial degree for the NLOS missile coast speed coefficient data array.

## 3.49.2.1 Initialization

The constant is initialized during execution of the CSU missile\_nlos\_init, called by CSC weapons\_init. Execution of the CSU missile\_nlos\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.49. - NLOS Missile Polynomial Degree Data Array for a summary of the constants data.

#define NLOS\_COAST\_SPEED\_DEG nlos\_miss\_poly\_deg[1]

## 3.49.2.2 Usage

During real-time execution, this constant is not recomputed. The maximum value for NLOS\_COAST\_SPEED\_DEG is 4, especially, the declared size of the nlos\_coast\_speed\_coeff array is 5.

# 3.49.2.2.1 Algorithm

NLOS\_COAST\_SPEED\_DEG is not used in the current calculations.

See APPENDIX J for a complete source code listing.

# 3.50 Nlos\_burn\_speed\_coeff

The nlos\_burn\_speed\_coeff array consists of the coefficients for a polynomial equation defining the NLOS missile burn speed with respect to time in the form using the Newton-Raphson method.

#### 3.50.1 Initialization

The nlos\_burn\_speed\_coeff array is initialized during execution of the CSU missile\_nlos\_init, called by CSC weapons\_init. Execution of the CSU missile\_nlos\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.50. - NLOS Missile Burn Speed Coefficient Data Array for a summary of the array data.

The following is the default declaration.

The array has a maximum size of 5 elements.

### 3.50.2 Usage

During real-time execution, this array is not recomputed. NLOS\_BURN\_SPEED\_DEG determines the number of elements of the array to be used in the polynomial evaluation.

## 3.50.2.1 Algorithm

The nlos\_burn\_speed\_coeff array is not used in the current calculations for NLOS missile burn speed. The NLOS missile speed profile is constant in phase one and phase two, using a time as the delimiter.

See APPENDIX J for a complete source code listing.

# 3.51 Nlos\_coast\_speed\_coeff

The nlos\_coast\_speed\_coeff array consists of the coefficients for a polynomial equation defining the NLOS missile coast speed with respect to time in the form using the Newton-Raphson method.

#### 3.51.1 Initialization

The nlos\_coast\_speed\_coeff array is initialized during execution of the CSU missile\_nlos\_init, called by CSC weapons\_init. Execution of the CSU missile\_nlos\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.51. - NLOS Missile Coast Speed Coefficient Data Array for a summary of the array data.

The following is the default declaration.

The array has a maximum size of 5 elements.

### 3.51.2 Usage

During real-time execution, this array is not recomputed. NLOS\_COAST\_SPEED\_DEG determines the number of elements of the array to be used in the polynomial evaluation.

## 3.51.2.1 Algorithm

The nlos\_coast\_speed\_coeff array is not used in the current calculations for NLOS missile coast speed. The NLOS missile speed profile is constant in phase one and phase two, using a time as the delimiter.

See APPENDIX J for a complete source code listing.

# 3.52 Hydra\_rkt\_char

The hydra\_rkt\_char array consists of characteristics and parameters describing a rocket launcher system and its performance constraints.

# 3.52.1 HYDRA\_LAUNCHER\_POS\_X

HYDRA\_LAUNCHER\_POS\_X is a constant defining the hydra launcher position in the x-axis.

#### 3.52.1.1 Initialization

The constant is initialized during execution of the CSU hydra\_init, called by CSC weapons\_init. Execution of the CSU hydra\_init is normally done only

once during CSCI initialization and is performed sequentially. See TABLE 5.1.52. - Hydra Rocket Configuration Data Array for a summary of the constant.

```
#define HYDRA_LAUNCHER_POS_X hydra_rkt_char[0]
```

### 3.52.1.2 Usage

During real-time execution, this constant is not recomputed.

## 3.52.1.2.1 Algorithm

HYDRA\_LAUNCHER\_POS\_X is used to initialize the left and right launcher position, rotational elements, and offset positions in the x-axis in a call to the CSU hydra\_init.

```
left_launcher_pos[0] = HYDRA_LAUNCHER_POS_X;
     right_launcher_pos[0] = HYDRA_LAUNCHER_POS_X;
     articulation_pos[1] = HYDRA_LAUNCHER_POS_Y;
     articulation_pos[2] = HYDRA_LAUNCHER_POS_Z;
   if(!rotate_init_element( &articulation_element, hull(),
             1.0, 0.0, 0.0, 0.0,
ARTICULATION_MIN,ARTICULATION_MAX,/*TWO_*/PI,/*rate*/
            0.0, HYDRA_LAUNCHER_POS_Y,
           HYDRA LAUNCHER POS_Z ))
     printf( "Rotate_Init_Element: articulation_element FAILED\n" );
  rotate_init_element( &pylon_L_element, articulation(), 0.0, 0.0, 1.0, 0.0,
             -TWO_PI, TWO_PI, TWO_PI, /*rate*/
            -HYDRA_LAUNCHER_POS_X, 0.0, 0.0);
  rotate_init_element( &pylon_R_element, articulation(), 0.0, 0.0, 1.0, 0.0,
             -TWO_PI, TWO_PI, TWO_PI, /*rate*/
            HYDRA_LAUNCHER_POS_X, 0.0, 0.0);
  missile_hydra_init( hydras, MAX_HYDRA70_ROCKET );
   missile_hydra_set_pylon_position_offsets(
                      HYDRA_LAUNCHER_POS_X,
                      HYDRA_LAUNCHER_POS_Y,
                      HYDRA_LAUNCHER_POS_Z );
```

### 3.52.2 HYDRA\_LAUNCHER\_POS\_Y

HYDRA\_LAUNCHER\_POS\_Y is a constant defining the hydra launcher position in the y-axis.

#### 3.52.2.1 Initialization

The constant is initialized during execution of the CSU hydra\_init, called by CSC weapons\_init. Execution of the CSU hydra\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.52. - Hydra Rocket Configuration Data Array for a summary of the constant.

```
#define HYDRA_LAUNCHER_POS_Y hydra_rkt_char[1]
```

## 3.52.2.2 Usage

During real-time execution, this constant is not recomputed.

## 3.52.2.2.1 Algorithm

HYDRA\_LAUNCHER\_POS\_Y is used to initialize the left and right launcher position, rotational elements, and offset positions in the y-axis in a call to the CSU hydra\_init.

```
rotate_init_element( &pylon_L_element, articulation(), 0.0, 0.0, 1.0, 0.0,

-TWO_PI, TWO_PI, TWO_PI, /*rate*/

-HYDRA_LAUNCHER_POS_X, 0.0, 0.0);

rotate_init_element( &pylon_R_element, articulation(), 0.0, 0.0, 1.0, 0.0,

-TWO_PI, TWO_PI, TWO_PI, /*rate*/

HYDRA_LAUNCHER_POS_X, 0.0, 0.0);

missile_hydra_init( hydras, MAX_HYDRA70_ROCKET );

missile_hydra_set_pylon_position_offsets(

HYDRA_LAUNCHER_POS_X,

HYDRA_LAUNCHER_POS_Z,

HYDRA_LAUNCHER_POS_Z);
```

## 3.52.3 HYDRA\_LAUNCHER\_POS\_Z

HYDRA\_LAUNCHER\_POS\_Z is a constant defining the hydra launcher position in the z-axis.

#### 3.52.3.1 Initialization

The constant is initialized during execution of the CSU hydra\_init, called by CSC weapons\_init. Execution of the CSU hydra\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.52. - Hydra Rocket Configuration Data Array for a summary of the constant.

```
#define HYDRA_LAUNCHER_POS_Z hydra_rkt_char[2]
```

#### 3.52.3.2 Usage

During real-time execution, this constant is not recomputed.

### 3.52.3.2.1 Algorithm

HYDRA\_LAUNCHER\_POS\_Z is used to initialize the left and right launcher position, rotational elements, and offset positions in the z-axis in a call to the CSU hydra\_init.

```
left_launcher_pos[0] = HYDRA_LAUNCHER_POS_X;
right_launcher_pos[0] = HYDRA_LAUNCHER_POS_X;
articulation_pos[1] = HYDRA_LAUNCHER_POS_Y;
articulation_pos[2] = HYDRA_LAUNCHER_POS_Z;
```

```
if(!rotate_init_element( &articulation_element, hull(),
             1.0, 0.0, 0.0, 0.0,
ARTICULATION_MIN,ARTICULATION_MAX,/*TWO_*/PI,/*rate*/
            0.0, HYDRA_LAUNCHER_POS_Y,
           HYDRA_LAUNCHER_POS_Z ))
     printf( "Rotate_Init_Element: articulation_element FAILED\n" );
  rotate_init_element( &pylon_L_element, articulation(), 0.0, 0.0, 1.0, 0.0,
             -TWO_PI, TWO_PI, TWO_PI, /*rate*/
             -HYDRA_LAUNCHER_POS_X, 0.0, 0.0);
  rotate_init_element( &pylon_R_element, articulation(), 0.0, 0.0, 1.0, 0.0,
             -TWO_PI, TWO_PI, TWO_PI, /*rate*/
             HYDRA_LAUNCHER_POS_X, 0.0, 0.0);
   missile_hydra_init( hydras, MAX_HYDRA70_ROCKET );
   missile_hydra_set_pylon_position_offsets(
                       HYDRA_LAUNCHER_POS_X,
                       HYDRA_LAUNCHER_POS_Y,
                       HYDRA_LAUNCHER_POS_Z );
```

# 3.52.4 SOVIET\_ARTICULATION

SOVIET\_ARTICULATION is a constant defining the angle of Soviet articulation in mils.

### 3.52.4.1 Initialization

The constant is initialized during execution of the CSU hydra\_init, called by CSC weapons\_init. Execution of the CSU hydra\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.52. - Hydra Rocket Configuration Data Array for a summary of the constant.

```
#define SOVIET_ARTICULATION ( mil_to_rad(hydra_rkt_char[3]))
```

## 3.52.4.2 Usage

During real-time execution, this constant is not recomputed.

### 3.52.4.2.1 Algorithm

SOVIET\_ARTICULATION is not used in the current calculations.

See APPENDIX N for a complete source code listing.

# 3.52.5 HULL\_NEG\_5\_PITCH

HULL\_NEG\_5\_PITCH is a constant defining the degrees of hull negative pitch.

### 3.52.5.1 Initialization

The constant is initialized during execution of the CSU hydra\_init, called by CSC weapons\_init. Execution of the CSU hydra\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.52. - Hydra Rocket Configuration Data Array for a summary of the constant.

```
#define HULL_NEG_5_PITCH ( deg_to_rad(hydra_rkt_char[4]))
```

## 3.52.5.2 Usage

During real-time execution, this constant is not recomputed.

# 3.52.5.2.1 Algorithm

HULL\_NEG\_5\_PITCH is used to compute the super elevation of the pylon articulation in the CSU hydra\_set\_pylon\_articulation.

```
super_elev += HULL_NEG_5_PITCH;
rotate_set_angle( articulation(), super_elev );
rotate_set_angle( pylon_R(), (- dispersion) );
rotate_set_angle( pylon_L(), dispersion );
```

### 3.52.6 ARTICULATION\_MAX

ARTICULATION\_MAX is a constant defining the degress of maximum articulation.

#### 3.52.6.1 Initialization

The constant is initialized during execution of the CSU hydra\_init, called by CSC weapons\_init. Execution of the CSU hydra\_init is normally done only once during CSCI initialization and is performed sequentially. See TABLE 5.1.52. - Hydra Rocket Configuration Data Array for a summary of the constant.

```
#define ARTICULATION_MAX ( deg_to_rad(hydra_rkt_char[5]))
```

#### 3.52.6.2 Usage

During real-time execution, this constant is not recomputed.

## 3.52.6.2.1 Algorithm

ARTICULATION\_MAX is used to limit the initialization of the rotation element in the call to the CSU hydra\_init.

```
if(!rotate_init_element( &articulation_element, hull(),
1.0, 0.0, 0.0, 0.0,
ARTICULATION_MIN,ARTICULATION_MAX,
/*TWO_*/PI,
/*rate*/
0.0, HYDRA_LAUNCHER_POS_Y,
HYDRA_LAUNCHER_POS_Z ))
```

See APPENDIX N for a complete source code listing.

#### 3.53.1 M151\_BURST\_SPREAD

M151\_BURST\_SPREAD is a constant defining the radius of the M151 burst spread, especially, the M151 is twin bursts 3 meters apart.

#### 3.53.1.1 Initialization

M151\_BURST\_SPREAD is initialized during execution of the CSU missile\_hydra\_init, called by CSU hydra\_init. See TABLE 5.1.53. - Hydra Rocket Characteristics Data Array for a summary of the constant. M151\_BURST\_SPREAD is also known as rkt\_hydra\_char[ 0].

The following declaration sets the default values.

```
static REAL rkt_hydra_char[12] =
                              /* twin bursts are 3 m apart */
 M151_BURST_SPREAD,
                              /* release submunitions 180 ft */
 M261_BURST_HEIGHT,
                              /* 0 m in front of target (49 ?) */
 M261 BURST RANGE,
 M261 BURST_SPREAD,
                              /* twin bursts are 13 m apart */
 M255 BURST_RANGE,
                              /* release darts 150 m front of tgt */
                              /* twin bursts are 35 m apart
 M255_BURST_SPREAD,
 FLECH 60 MAX RANGE,
                              /* darts fly total of 750 m
 50.0,
                 /* hydra minimum range */
 5000.0,
                 /* hydra maximum range for Soviet S-5 57mm Rocket */
                 /* hydra maximum range for _M151 [actual 9000 m] */
 7000.0,
                 /* hydra maximum range for M261 */
 7000.0,
                 /* hydra maximum range for M255 */
 3200.0
```

## 3.53.1.2 Usage

During real-time execution, this constant is not recomputed.

## 3.53.1.2.1 Algorithm

Rkt\_hydra\_char[0] is used to compute the lead\_angle when the type of rocket is HE with 10 LB warhead by a call to the CSU missile\_hydra\_set\_pylon\_articulation.

# 3.53.2 M261\_BURST\_HEIGHT

M261\_BURST\_HEIGHT is a constant defining the height of release for the M261 burst, especially, release of submunitions at 180 feet above the ground level.

### 3.53.2.1 Initialization

M261\_BURST\_HEIGHT is initialized during execution of the CSU missile\_hydra\_init, called by CSU hydra\_init. See TABLE 5.1.53. - Hydra Rocket Characteristics Data Array for a summary of the constant. M261\_BURST\_HEIGHT is also known as rkt\_hydra\_char[ 1].

The following declaration sets the default values.

```
static REAL rkt_hydra_char[12] =
                              /* twin bursts are 3 m apart */
 M151_BURST_SPREAD,
 M261_BURST_HEIGHT,
                              /* release submunitions 180 ft */
                              /* 0 m in front of target (49 ?) */
 M261_BURST_RANGE,
                              /* twin bursts are 13 m apart */
 M261 BURST SPREAD,
                              /* release darts 150 m front of tgt */
 M255 BURST RANGE,
                             · /* twin bursts are 35 m apart
 M255 BURST_SPREAD,
 FLECH_60_MAX_RANGE,
                              /* darts fly total of 750 m
                  /* hydra minimum range */
  50.0,
                  /* hydra maximum range for Soviet S-5 57mm Rocket */
 5000.0,
                 /* hydra maximum range for _M151 [actual 9000 m] */
 7000.0,
                 /* hydra maximum range for M261 */
 7000.0,
                  /* hydra maximum range for M255 */
 3200.0
```

### 3.53.2.2 Usage

During real-time execution, this constant is not recomputed.

## 3.53.2.2.1 Algorithm

Rkt\_hydra\_char[ 1] is used to compute ballistics trajectory when the rocket is a type MPSM by a call to the CSU missile\_hydra\_set\_pylon\_articulation.

Rkt\_hydra\_char[ 1] is assigned to submunition impact height when the rocket is a type MPSM by a call to the CSU missile\_hydra\_fire.

```
case ROCKET_MPSM: /* Multi-Purpose Sub-Munition */
bmptr->max_range = HYDRA_MAX_RANGE_M261;
rkt->sub_mun_type = SUB_MUN_IMPACT;
rkt->sub_ammo_type = munition_US_M73;
rkt->sub_munition.impact.ammo = munition_US_M73;
rkt->sub_munition.impact.fuze = munition_US_M433;
rkt->sub_munition.impact.quantity = m73_per_m261_burst;
rkt->sub_munition.impact.height = rkt_hydra_char[ 1];
fuze = munition_US_M439;
break;
```

See APPENDIX M for a complete source code listing.

# 3.53.3 M261\_BURST\_RANGE

M261\_BURST\_RANGE is a constant defining the distance from the target to the burst, especially, for the M261 burst at 0 meters in front of target.

### 3.53.3.1 Initialization

M261\_BURST\_RANGE is initialized during execution of the CSU missile\_hydra\_init, called by CSU hydra\_init. See TABLE 5.1.53. - Hydra Rocket Characteristics Data Array for a summary of the constant. M261\_BURST\_RANGE is also known as rkt\_hydra\_char[2].

The following declaration sets the default values.

```
static REAL rkt_hydra_char[12] =
                             /* twin bursts are 3 m apart */
 M151_BURST_SPREAD,
                             /* release submunitions 180 ft */
 M261 BURST HEIGHT,
 M261_BURST_RANGE,
                             /* 0 m in front of target (49 ?) */
 M261 BURST_SPREAD,
                             /* twin bursts are 13 m apart */
 M255 BURST RANGE,
                             /* release darts 150 m front of tgt */
 M255 BURST_SPREAD,
                             · /* twin bursts are 35 m apart
 FLECH_60_MAX_RANGE,
                             /* darts fly total of 750 m
                 /* hydra minimum range */
 50.0,
5000.0,
                 /* hydra maximum range for Soviet S-5 57mm Rocket */
7000.0,
                 /* hydra maximum range for _M151 [actual 9000 m] */
                 /* hydra maximum range for M261 */
7000.0,
                 /* hydra maximum range for M255 */
3200.0
```

### 3.53.3.2 Usage

During real-time execution, this constant is not recomputed.

## 3.53.3.2.1 Algorithm

Rkt\_hydra\_char[2] is not used in the current calculations.

See APPENDIX M for a complete source code listing.

### 3.53.4 M261\_BURST\_SPREAD

M261\_BURST\_SPREAD is a constant defining the radius of the M261 burst spread, especially, the M261 is twin bursts 13 meters apart.

#### 3.53.4.1 Initialization

M261\_BURST\_SPREAD array is initialized during execution of the CSU missile\_hydra\_init, called by CSU hydra\_init. See TABLE 5.1.53. - Hydra Rocket Characteristics Data Array for a summary of the constant. M261\_BURST\_SPREAD is also known as rkt\_hydra\_char[ 3].

The following declaration sets the default values.

```
static REAL rkt_hydra_char[12] =
                              /* twin bursts are 3 m apart */
 M151_BURST_SPREAD,
                             /* release submunitions 180 ft */
 M261 BURST HEIGHT,
 M261 BURST_RANGE,
                             /* 0 m in front of target (49 ?) */
 M261 BURST_SPREAD,
                             /* twin bursts are 13 m apart */
 M255 BURST_RANGE,
                             /* release darts 150 m front of tgt */
                             ·/* twin bursts are 35 m apart
 M255 BURST_SPREAD,
 FLECH_60_MAX_RANGE,
                             /* darts fly total of 750 m
                 /* hydra minimum range */
 50.0,
                  /* hydra maximum range for Soviet S-5 57mm Rocket */
 5000.0,
                 /* hydra maximum range for _M151 [actual 9000 m] */
7000.0,
                 /* hydra maximum range for M261 */
7000.0,
                 /* hydra maximum range for M255 */
3200.0
```

### 3.53.4.2 Usage

During real-time execution, this constant is not recomputed.

## 3.53.4.2.1 Algorithm

Rkt\_hydra\_char[3] is used to compute the lead angle when the rocket is a type MPSM by a call to the CSU missile\_hydra\_set\_pylon\_articulation.

See APPENDIX M for a complete source code listing.

#### 3.53.5 M255\_BURST\_RANGE

M255\_BURST\_RANGE is a constant defining the distance from the target to the burst, especially, for the M255 burst at 150 meters in front of target.

#### 3.53.5.1 Initialization

M255\_BURST\_RANGE is initialized during execution of the CSU missile\_hydra\_init, called by CSU hydra\_init. See TABLE 5.1.53. - Hydra Rocket Characteristics Data Array for a summary of the constant. M255\_BURST\_RANGE is also known as rkt\_hydra\_char[4].

The following declaration sets the default values.

```
static REAL rkt_hydra_char[12] =
                             /* twin bursts are 3 m apart */
 M151_BURST_SPREAD,
                             /* release submunitions 180 ft */
 M261 BURST HEIGHT,
 M261 BURST_RANGE,
                             /* 0 m in front of target (49 ?) */
 M261_BURST_SPREAD,
                             /* twin bursts are 13 m apart */
 M255 BURST_RANGE,
                             /* release darts 150 m front of tgt */
                             /* twin bursts are 35 m apart
 M255_BURST_SPREAD,
 FLECH 60 MAX_RANGE,
                             /* darts fly total of 750 m
 50.0,
                 /* hydra minimum range */
5000.0,
                 /* hydra maximum range for Soviet S-5 57mm Rocket */
7000.0,
                 /* hydra maximum range for _M151 [actual 9000 m] */
7000.0,
                 /* hydra maximum range for M261 */
3200.0
                 /* hydra maximum range for M255 */
```

#### 3.53.5.2 Usage

During real-time execution, this constant is not recomputed.

### 3.53.5.2.1 Algorithm

Rkt\_hydra\_char[4] is used to compute ballistics trajectory and lead angle when the rocket is a type FLECHETTE by a call to the CSU missile\_hydra\_set\_pylon\_articulation.

## 3.53.6 M255\_BURST\_SPREAD

M255\_BURST\_SPREAD is a constant defining the radius of the M255 burst spread, especially, the M255 is twin bursts 35 meters apart.

#### 3.53.6.1 Initialization

M255\_BURST\_SPREAD is initialized during execution of the CSU missile\_hydra\_init, called by CSU hydra\_init. See TABLE 5.1.53. - Hydra Rocket Characteristics Data Array for a summary of the constant. M255\_BURST\_SPREAD is also known as rkt\_hydra\_char[5].

The following declaration sets the default values.

```
static REAL rkt_hydra_char[12] =
  M151 BURST_SPREAD,
                             /* twin bursts are 3 m apart */
                              /* release submunitions 180 ft */
 M261 BURST HEIGHT,
                              /* 0 m in front of target (49 ?) */
  M261 BURST_RANGE,
                              /* twin bursts are 13 m apart */
  M261 BURST SPREAD,
                              /* release darts 150 m front of tgt */
  M255_BURST_RANGE,
                             ./* twin bursts are 35 m apart
  M255 BURST_SPREAD,
 FLECH_60_MAX_RANGE,
                             /* darts fly total of 750 m
                  /* hydra minimum range */
  50.0,
                 /* hydra maximum range for Soviet S-5 57mm Rocket */
 5000.0,
                 /* hydra maximum range for _M151 [actual 9000 m] */
 7000.0,
                 /* hydra maximum range for M261 */
 7000.0,
                 /* hydra maximum range for M255 */
 3200.0
```

## 3.53.6.2 Usage

During real-time execution, this constant is not recomputed.

## 3.53.6.2.1 Algorithm

Rkt\_hydra\_char[5] is used to compute ballistics trajectory and lead angle when the rocket is a type FLECHETTE by a call to the CSU missile hydra\_set\_pylon\_articulation.

See APPENDIX M for a complete source code listing.

### 3.53.7 FLECH\_60\_MAX\_RANGE

FLECH\_60\_MAX\_RANGE is a constant defining the total distance the darts fly in meters after the proximity fuze detonates. At the maximum range, the flechette rounds have lost the momentum and fall to the ground.

#### 3.53.7.1 Initialization

FLECH\_60\_MAX\_RANGE is initialized during execution of the CSU missile\_hydra\_init, called by CSU hydra\_init. See TABLE 5.1.53. - Hydra Rocket Characteristics Data Array for a summary of the constant. FLECH\_60\_MAX\_RANGE is also known as rkt\_hydra\_char[6].

The following declaration sets the default values.

```
static REAL rkt_hydra_char[12] =
 M151 BURST_SPREAD,
                             /* twin bursts are 3 m apart */
 M261_BURST_HEIGHT,
                             /* release submunitions 180 ft */
 M261_BURST_RANGE,
                             /* 0 m in front of target (49 ?) */
 M261_BURST_SPREAD,
                             /* twin bursts are 13 m apart */
 M255_BURST_RANGE,
                             /* release darts 150 m front of tgt */
 M255_BURST_SPREAD,
                             /* twin bursts are 35 m apart
 FLECH_60_MAX_RANGE,
                             /* darts fly total of 750 m
                 /* hydra minimum range */
 50.0,
                 /* hydra maximum range for Soviet S-5 57mm Rocket */
 5000.0,
 7000.0,
                 /* hydra maximum range for _M151 [actual 9000 m] */
 7000.0,
                 /* hydra maximum range for M261 */
                 /* hydra maximum range for M255 */
 3200.0
```

## 3.53.7.2 Usage

During real-time execution, this constant is not recomputed.

# 3.53.7.2.1 Algorithm

FLECH\_60\_MAX\_RANGE is not used in the current calculations.

See APPENDIX M for a complete source code listing.

## 3.53.8 HYDRA\_MIN\_RANGE

HYDRA\_MIN\_RANGE is a constant defining the hydra minimum range.

### 3.53.8.1 Initialization

HYDRA\_MIN\_RANGE is initialized during execution of the CSU missile\_hydra\_init, called by CSU hydra\_init. See TABLE 5.1.53. - Hydra Rocket Characteristics Data Array for a summary of the constant.

```
#define HYDRA_MIN_RANGE rkt_hydra_char[ 7]
```

The following declaration sets the default values.

```
static REAL rkt_hydra_char[12] =
                              /* twin bursts are 3 m apart */
 M151_BURST_SPREAD,
                              /* release submunitions 180 ft */
 M261 BURST_HEIGHT,
                              /* 0 m in front of target (49 ?) */
  M261_BURST_RANGE,
                              /* twin bursts are 13 m apart */
  M261_BURST_SPREAD,
                              /* release darts 150 m front of tgt */
 ' M255_BURST_RANGE,
                              /* twin bursts are 35 m apart
  M255_BURST_SPREAD,
                              /* darts fly total of 750 m
  FLECH_60_MAX_RANGE,
                  /* hydra minimum range */
  50.0,
                  /* hydra maximum range for Soviet S-5 57mm Rocket */
 5000.0,
                /* hydra maximum range for _M151 [actual 9000 m] */
 7000.0,
                  /* hydra maximum range for M261 */
 7000.0,
                  /* hydra maximum range for M255 */
 3200.0
};
```

### 3.53.8.2 Usage

During real-time execution, this constant is not recomputed.

### 3.53.8.2.1 Algorithm

HYDRA\_MIN\_RANGE is used to limit the range to the target by a call to the CSU missile\_hydra-set\_pylon\_articulation.

```
if( tgt_range < HYDRA_MIN_RANGE )
    range = HYDRA_MIN_RANGE;
else if(( max_range_limit > 0.0 ) &&
        ( tgt_range > max_range_limit ) )
    range = max_range_limit;
```

```
else
range = tgt_range;
```

See APPENDIX M for a complete source code listing.

### 3.53.9 HYDRA\_MAX\_RANGE\_S5

HYDRA\_MAX\_RANGE\_S5 is a constant defining the hydra maximum range for Soviet S-5 57mm rocket.

### 3.53.9.1 Initialization

HYDRA\_MAX\_RANGE\_S5 is initialized during execution of the CSU missile\_hydra\_init, called by CSU hydra\_init. See TABLE 5.1.53. - Hydra Rocket Characteristics Data Array for a summary of the constant.

```
#define HYDRA_MAX_RANGE_S5 rkt_hydra_char[ 8]
```

The following declaration sets the default values.

```
static REAL rkt_hydra_char[12] =
 M151_BURST_SPREAD,
                              /* twin bursts are 3 m apart */
 M261 BURST_HEIGHT,
                              /* release submunitions 180 ft */
 M261_BURST_RANGE,
                              /* 0 m in front of target (49 ?) */
 M261_BURST_SPREAD,
                              /* twin bursts are 13 m apart */
                              /* release darts 150 m front of tgt */
 M255 BURST RANGE,
 M255_BURST_SPREAD,
                              /* twin bursts are 35 m apart
 FLECH 60 MAX RANGE,
                              /* darts fly total of 750 m
 50.0,
                 /* hydra minimum range */
                 /* hydra maximum range for Soviet S-5 57mm Rocket */
5000.0,
7000.0,
                 /* hydra maximum range for _M151 [actual 9000 m] */
                 /* hydra maximum range for M261 */
7000.0,
3200.0
                 /* hydra maximum range for M255 */
};
```

### 3.53.9.2 Usage

During real-time execution, this constant is not recomputed.

### 3.53.9.2.1 Algorithm

HYDRA\_MAX\_RANGE\_S5 is not used in the current calculations.

See APPENDIX M for a complete source code listing.

### 3.53.10 HYDRA\_MAX\_RANGE\_M151

HYDRA\_MAX\_RANGE\_M151 is a constant defining the hydra maximum range for the M151.

### 3.53.10.1 Initialization

HYDRA\_MAX\_RANGE\_M151 array is initialized during execution of the CSU missile\_hydra\_init, called by CSU hydra\_init. See TABLE 5.1.53. - Hydra Rocket Characteristics Data Array for a summary of the constant.

```
#define HYDRA_MAX_RANGE_M151 rkt_hydra_char[9]
```

The following declaration sets the default values.

```
static REAL rkt_hydra_char[12] =
                              /* twin bursts are 3 m apart */
 M151 BURST_SPREAD,
                              /* release submunitions 180 ft */
 M261_BURST_HEIGHT,
                              /* 0 m in front of target (49 ?) */
  M261_BURST_RANGE,
                              /* twin bursts are 13 m apart
 M261 BURST_SPREAD,
                              /* release darts 150 m front of tgt */
  M255_BURST_RANGE,
  M255_BURST_SPREAD,
                              /* twin bursts are 35 m apart
                              /* darts fly total of 750 m
 FLECH 60 MAX_RANGE,
                  /* hydra minimum range */
 50.0,
                  /* hydra maximum range for Soviet S-5 57mm Rocket */
 5000.0,
                 /* hydra maximum range for _M151 [actual 9000 m] */
 7000.0,
                 /* hydra maximum range for M261 */
 7000.0,
                 /* hydra maximum range for M255 */
 3200.0
```

### 3.53.10.2 Usage

During real-time execution, this constant is not recomputed.

### 3.53.10.2.1 Algorithm

HYDRA\_MAX\_RANGE\_M151 is used to bound the limit of the range for the individual rocket when the rocket type is HE by a call to the CSU missile\_hydra\_set\_pylon\_articulation.

HYDRA\_MAX\_RANGE\_M151 is assigned to the maximum range variable for the individual rocket when the rocket type is HE by a call to the CSU missile\_hydra\_fire.

See APPENDIX M for a complete source code listing.

### 3.53.11 HYDRA\_MAX\_RANGE\_M261

HYDRA\_MAX\_RANGE\_M261 is a constant defining the hydra maximum range for the M261.

### 3.53.11.1 Initialization

HYDRA\_MAX\_RANGE\_M261 is initialized during execution of the CSU missile\_hydra\_init, called by CSU hydra\_init. See TABLE 5.1.53. - Hydra Rocket Characteristics Data Array for a summary of the constant.

```
#define HYDRA_MAX_RANGE_M261 rkt_hydra_char[10]
```

The following declaration sets the default values.

```
static REAL rkt_hydra_char[12] =
                              /* twin bursts are 3 m apart */
 M151 BURST SPREAD,
                             /* release submunitions 180 ft */
 M261_BURST_HEIGHT,
 M261_BURST_RANGE,
                              /* 0 m in front of target (49 ?) */.
 M261 BURST_SPREAD,
                              /* twin bursts are 13 m apart */
                              /* release darts 150 m front of tgt */
 M255_BURST_RANGE,
 M255 BURST SPREAD,
                              /* twin bursts are 35 m apart
 FLECH_60_MAX_RANGE,
                              /* darts fly total of 750 m
                 /* hydra minimum range */
 50.0,
                 /* hydra maximum range for Soviet S-5 57mm Rocket */
 5000.0,
 7000.0,
                 /* hydra maximum range for _M151 [actual 9000 m] */
                 /* hydra maximum range for M261 */
 7000.0,
3200.0
                 /* hydra maximum range for M255 */
```

### 3.53.11.2 Usage

During real-time execution, this constant is not recomputed.

### 3.53.11.2.1 Algorithm

HYDRA\_MAX\_RANGE\_M261 is used to bound the limit of the range for the individual rocket when the rocket type is MPSM by a call to the CSU missile\_hydra\_set\_pylon\_articulation.

HYDRA\_MAX\_RANGE\_M261 is assigned to the maximum range variable for the individual rocket when the rocket type is MPSM by a call to the CSU missile\_hydra\_fire.

```
case ROCKET_MPSM: /* Multi-Purpose Sub-Munition */
bmptr->max_range = HYDRA_MAX_RANGE_M261;
rkt->sub_mun_type = SUB_MUN_IMPACT;
rkt->sub_ammo_type = munition_US_M73;
rkt->sub_munition.impact.ammo = munition_US_M73;
rkt->sub_munition.impact.fuze = munition_US_M433;
rkt->sub_munition.impact.quantity = m73_per_m261_burst;
rkt->sub_munition.impact.height = rkt_hydra_char[ 1];
fuze = munition_US_M439;
break;
```

See APPENDIX M for a complete source code listing.

### 3.53.12 HYDRA\_MAX\_RANGE\_M255

HYDRA\_MAX\_RANGE\_M255 is a constant defining the hydra maximum range for the M255.

### 3.53.12.1 Initialization

HYDRA\_MAX\_RANGE\_M255 is initialized during execution of the CSU missile\_hydra\_init, called by CSU hydra\_init. See TABLE 5.1.53. - Hydra Rocket Characteristics Data Array for a summary of the constant.

#define HYDRA\_MAX\_RANGE\_M255 rkt\_hydra\_char[11]

The following declaration sets the default values.

```
static REAL rkt_hydra_char[12] =
 M151_BURST_SPREAD,
                              /* twin bursts are 3 m apart */
                              /* release submunitions 180 ft */
 M261_BURST_HEIGHT,
                              /* 0 m in front of target (49 ?) */
 M261_BURST_RANGE,
 M261_BURST_SPREAD,
                              /* twin bursts are 13 m apart */
 M255 BURST_RANGE,
                              /* release darts 150 m front of tgt */
 M255_BURST_SPREAD,
                             ·/* twin bursts are 35 m apart
 FLECH_60_MAX_RANGE,
                              /* darts fly total of 750 m
                 /* hydra minimum range */
 50.0,
                  /* hydra maximum range for Soviet S-5 57mm Rocket */
 5000.0,
 7000.0,
                 /* hydra maximum range for _M151 [actual 9000 m] */
 7000.0,
                 /* hydra maximum range for M261 */
                  /* hydra maximum range for M255 */
3200.0
```

### 3.53.12.2 Usage

During real-time execution, this constant is not recomputed.

### 3.53.12.2.1 Algorithm

HYDRA\_MAX\_RANGE\_M255 is used to bound the limit of the range for the individual rocket when the rocket type is FLECHETTE by a call to the CSU missile\_hydra\_set\_pylon\_articulation.

HYDRA\_MAX\_RANGE\_M255 is assigned to the maximum range variable for the individual rocket when the rocket type is FLECHETTE by a call to the CSU missile\_hydra\_fire.

```
case ROCKET_FLECHETTE: /* Flechette discharging warhead */
bmptr->max_range = HYDRA_MAX_RANGE_M255;
rkt->sub_mun_type = SUB_MUN_CANISTER;
rkt->sub_ammo_type = munition_US_Flechette_60;
rkt->sub_munition.dart.ammo = munition_US_Flechette_60;
rkt->sub_munition.dart.fuze = 0;
fuze = munition_US_M439;
break;
```

See APPENDIX M for a complete source code listing.

### 3.54 Sub\_m73\_char

The sub\_m73\_char array consists of characteristics and parameters describing a M73 submunition flyout.

### 3.54.1 Sub\_m73\_char[ 0]

Sub\_m73\_char[0] is a constant defining the downward acceleration of gravity as 75% of gravity, per tick squared (75% \* (9.8m/sec\*\*2)/225 ticks\*\*2).

### 3.54.1.1 Initialization

The sub\_m73\_char[0] constant is initialized during execution of the CSU missile\_m73\_init, called by CSU missile\_hydra\_fly\_rockets. See TABLE 5.1.54. - Submunitions M73 Characteristics Data Array for a summary of the array data.

The following declaration is for the default values.

### 3.54.1.2 Usage

During real-time execution, this constant is not recomputed.

### 3.54.1.2.1 Algorithm

Sub\_m73\_char[0] is used to compute the impact timer for the for the M73 by a call to the CSU missile\_m73\_drop.

```
impact = &(sub_mun->impact);
if( impact->timer == 0 )
    if( missile_util_comm_check_sub_mun( bmptr,
                                    MSL TYPE BALLISTIC,
                                    sub_mun, SUB_MUN_IMPACT ))
    if(impact->distance > 0.0)
       impact->timer = (int)
         ((8 * scaled rand()) + 1.0 +
          (sqrt((1.9 * impact->distance) / sub_m73_char[0])));
    else
       impact->timer = -1;
, }
  else
     impact_pt[X] = bmptr->location[X];
     impact_pt[Y] = bmptr->location[Y] - 10;
     if(traj_up)
       impact_pt[Z] = bmptr->location[Z] + impact->distance;
     else
       impact_pt[Z] = 10;
     traj_up = (! traj_up);
       missile_util_comm_release_sub_munition( bmptr,
                                    MSL_TYPE_BALLISTIC,
                                    sub_mun, SUB_MUN_IMPACT,
                                    impact_pt, zero_velocity );
   return(FALSE);
else
   if( bmptr->time < impact->timer )
                                       /* wait until sub_mun's */
                        /* hit the ground.... */
                                 /* incr time counter */
     bmptr->time += 1;
     return(FALSE);
  }
                          /* ie. time == timer */
   else
```

See APPENDIX O for a complete source code listing.

### 3.54.2 M73\_FOOT\_ANGLE\_X

M73\_FOOT\_ANGLE\_X is a constant defining the dispersion angle on the x-axis of bomblettes as they fall, especially, falling with +/- 8.8 degrees angular displacement along the x-axis.

### 3.54.2.1 Initialization

M73\_FOOT\_ANGLE\_X is initialized during execution of the CSU missile\_m73\_init, called by CSU missile\_hydra\_fly\_rockets. See TABLE 5.1.54. - Submunitions M73 Characteristics Data Array for a summary of the constant. M73\_FOOT\_ANGLE\_X is also known as sub\_m73\_char[2].

The following declaration is for the default values.

### 3.54.2.2 Usage

During real-time execution, this constant is not recomputed.

### 3.54.2.2.1 Algorithm

Sub\_m73\_char[2] is used to compute the detonation point in CSU missile\_m73\_get\_impact.

```
 x = \text{height } * \sin(\deg_{to}_{rad}( sub_{m73}_{char}[1] * (0.50 - scaled_{rand}())));   y = \text{height } * \sin(\deg_{to}_{rad}( sub_{m73}_{char}[2] * (0.50 - scaled_{rand}())));   \det(X) = x * mCw[0][0] - y * mCw[0][1];   \det(X) = y * mCw[0][0] + x * mCw[0][1];   \det(X) = x * mCw[0][0] + x * mCw[0][1];   \det(X) = x * mCw[0][0] + x * mCw[0][1];   \det(X) = x * mCw[0][0] + x * mCw[0][1];   \det(X) = x * mCw[0][0] + x * mCw[0][1];   \det(X) = x * mCw[0][0] + x * mCw[0][1];   \det(X) = x * mCw[0][0] + x * mCw[0][1];   \det(X) = x * mCw[0][0] + x * mCw[0][1];   \det(X) = x * mCw[0][0] + x * mCw[0][1];   \det(X) = x * mCw[0][0] + x * mCw[0][1];   \det(X) = x * mCw[0][0] + x * mCw[0][1];   \det(X) = x * mCw[0][0] + x * mCw[0][0][0] + x * mCw[0][0
```

See APPENDIX O for a complete source code listing.

### 3.54.3 M73\_FOOT\_ANGLE\_Y

M73\_FOOT\_ANGLE\_Y is a constant defining the dispersion angle on the y-axis of bomblettes as they fall, especially, falling with +/- 12.35 degrees angular displacement along the y-axis.

### 3.54.3.1 Initialization

M73\_FOOT\_ANGLE\_Y is initialized during execution of the CSU missile\_m73\_init, called by CSU missile\_hydra\_fly\_rockets. See TABLE 5.1.54. - Submunitions M73 Characteristics Data Array for a summary of the constant. M73\_FOOT\_ANGLE\_Y is also known as sub\_m73\_char[3].

The following declaration is for the default values.

### 3.54.3.2 Usage

During real-time execution, this constant is not recomputed.

### 3.54.3.2.1 Algorithm

Sub\_m73\_char[3] is used to compute the detonation point in CSU missile\_m73\_get\_impact.

See APPENDIX O for a complete source code listing.

### 3.55 Sub\_flech\_char

The sub\_flech\_char array consists of characteristics and parameters describing M73 bomblettes falling.

### 3.55.1 INVEST\_DIST\_SQ

The INVEST\_DIST\_SQ is a constant defining the area at a maximum speed of less than 100 m/sec.

### 3.55.1.1 Initialization

The INVEST\_DIST\_SQ is initialized during execution of the CSU missile\_flechette\_init, called by CSU missile\_hydra\_fly\_rockets. See TABLE 5.1.55. - Submunitions Flechette Characteristics Data Array for a summary of the constant.

#define INVEST\_DIST\_SQ sub\_flech\_char[0]

The following declaration is for the default values.

### 3.55.1.2 Usage

During real-time execution, this constant is not recomputed.

### 3.55.1.2.1 Algorithm

INVEST\_DIST\_SQ is used to compute detonation of the proximity fuze by a call in the CSU missile\_flechette\_fly.

```
* PROX_FUZE */
  if( missile fuze_all_prox(.bmptr,
                MSL_TYPE_BALLISTIC, PROX_FUZE_ON_ALL_VEH,
               &(null_VehicleID), &(dart->pptr),
                veh_list, INVEST_DIST_SQ, FUZE_DIST_SQ))
   do
/* DETONATION ? */
       if( missile_util_comm_check_sub_mun( bmptr,
                      MSL_TYPE_BALLISTIC,
                      sub_mun, SUB_MUN_CANISTER ))
          missile_util_comm_release_sub_munition( bmptr,
                           MSL_TYPE_BALLISTIC,
                           sub mun,
                            SUB_MUN_CANISTER,
                           zero_vector,
                          velocity);
    } while( dart->pptr != NULL &&
         missile_fuze_detonate_prox( bmptr, MSL_TYPE_BALLISTIC,
                     &(dart->pptr), FUZE_DIST_SQ, 0 ));
```

See APPENDIX P for a complete source code listing.

### 3.55.2 FUZE\_DIST\_SQ

FUZE\_DIST\_SQ is a constant defining the square of the radius of the cylinder describing the flechettes fly in a cylinder with a radius of 17.5 meters and a length of 750 meters

### 3.55.2.1 Initialization

The FUZE\_DIST\_SQ is initialized during execution of the CSU missile\_flechette\_init, called by CSU missile\_hydra\_fly\_rockets. See TABLE 5.1.55. - Submunitions Flechette Characteristics Data Array for a summary of the constant.

```
#define FUZE_DIST_SQ sub_flech_char[1]
```

The following declaration is for the default values.

### 3.55.2.2 Usage

During real-time execution, this constant is not recomputed.

### 3.55.2.2.1 Algorithm

FUZE\_DIST\_SQ is used to compute detonation of the proximity fuze by a call in the CSU missile\_flechette\_fly.

```
* PROX FUZE */
  if( missile_fuze_all_prox( bmptr,
                MSL TYPE BALLISTIC, PROX_FUZE_ON_ALL_VEH,
               &(null VehicleID), &(dart->pptr),
                veh list, INVEST_DIST_SQ, FUZE_DIST_SQ))
   do
/* DETONATION ? */
       if( missile_util_comm_check_sub_mun( bmptr,
                      MSL TYPE BALLISTIC,
                      sub_mun, SUB_MUN_CANISTER ))
          missile util_comm_release_sub_munition( bmptr,
                          MSL_TYPE_BALLISTIC,
                          sub mun,
                           SUB MUN_CANISTER,
                          zero_vector,
                          velocity);
   } while( dart->pptr != NULL &&
         missile_fuze_detonate_prox( bmptr, MSL_TYPE_BALLISTIC,
                     &(dart->pptr), FUZE_DIST_SQ, 0 ));
```

See APPENDIX P for a complete source code listing.

### 3.55.3 FLECH\_60\_MAX\_RANGE

FLECH\_60\_MAX\_RANGE is a constant defining the total distance the darts fly in meters after the proximity fuze detonates. At the maximum range, the flechette rounds have lost the momentum and fall to the ground. This constant is also known as sub\_flech\_char[2].

### 3.55.3.1 Initialization

The constant FLECH\_60\_MAX\_RANGE is initialized during execution of the CSU missile\_flechette\_init, called by CSU missile\_hydra\_fly\_rockets. See TABLE 5.1.55. - Submunitions Flechette Characteristics Data Array for a summary of the constant.

The following declaration is for the default values.

### 3.55.3.2 Usage

During real-time execution, this constant is not recomputed.

### 3.55.3.2.1 Algorithm

FLECH\_60\_MAX\_RANGE is used to compute the termination of the flight for the canister and darts.

```
dart->distance += bmptr->speed;
if( dart->distance >= sub_flech_char[2] )
return( FALSE );
```

See APPENDIX P for a complete source code listing.

### 3.56 Flechette\_speed\_coef

The flechette\_speed\_coef array consists of the coefficients for a polynomial equation defining the flechette flyout speed with respect to time in the form using the Newton-Raphson method.

### 3.56.1 Initialization

The flechette\_speed\_coef array is initialized during execution of the CSU missile\_flechette\_init, called by CSU missile\_hydra\_fly\_rockets. See TABLE 5.1.56. - Flechette Speed Data Array for a summary of the array data.

The array has a maximum size of 5 elements.

### 3.56.2 Usage

During real-time execution, this array is not recomputed. FLECHETTE\_SPEED\_DEG determines the number of elements of the array to be used in the polynomial evaluation.

### 3.56.2.1 Algorithm

The flechette\_speed\_coef array is used to compute the flechette speed during free fall using the CSU missile\_util\_eval\_poly, and called by the CSU missile\_flechette\_fly. The CSU missile\_util\_eval\_poly uses the Newton-Raphson method to evaluate the polynomial with inputs of degree of polynomial, coefficient array, and distance.

See APPENDIX P for a complete source code listing.

### 4. Error messages.

The error messages are located in the source code. See the appropriate Appendix for the error messages.

### 5. CSCI data.

This section describes only those global data elements modified or added within the CSCI under this delivery order. For ease in readability and maintenance, the information is provided in tables.

### 5.1. Data elements internal to the CSCI.

a. For data elements internal to the CSCI, the following tables describe the data arrays and the data.

## TABLE 5.1. - SUMMARY of DATA ARRAYS

NAME of DATA ARRAY	DESCRIPTION [NOTE 1]	SIZE of ARRAY	DATA TYPE [HOTE 2]	FREQUENCY of	DECLARATION/DEFAULT MODULE	DATA SOURCE
sero_data	This data array condute of characteristics and parameters describing the physical vehicle and its aerodynamic performance and control.	821	REAL	15 112	rwa_aerodyn.c	simnet/data/rwa_aero.d
sero_irit	This data array consists of initial values for positions of the control inputs, stabilator augmentation integrators, attitude control integrators, and hover aumentation integrators, and hover aumentation interators.	20	REAL	15 Hz	rwa_aerodyn.c	ainnet/data/rw_ae_in.d
eero_simple	This data array consists of characteristics and parameters describing the physical vehicle and its serodynamic performance and control in the simple mode.	20	REAL	15 Hz	rwa_aerodyn.c	strunet/data/rw_ae_sp.d
aero_stealth	This data array consists of characteristics and parameters describing the physical vehicle and its aerodynamic performance and control in the "stealth" mode.	50	REAL	15 Hz	rwa_aerodyn.c	simnet/data/rw_ae_std
engine_data		. 20	REAL	15 FIz	rwa_engine.c	sinnel/dala/rwa_engn.d
engine_init_data	This data array consists of initial values of the current engine state, ive formance, and control.	01	REAL	15 Hz	rwa_engine.c	simnet/data/rw_en_in.d
engine_stat_data	This data array consists of the initial values for flight time, engine status, number of engines, and nowerizain damage status.	10	भ	15 Hz	rwa_engine.c	simnet/data/rw_en_st.d
kinemal_data	This data array consists of kinematic constants and limits for the vehicle and its control.	20	REAL	15 Hz	rwa_kinemat.c	simnet/data/rwa_kine.d
kinemat_init_data	This data array consists of initial values for kinematic variables including velocity, angle-of-attack, pitch, allitude, heading, and g-force.	8	REAL	15 Hz	rwa_kinemal.c	simnet/data/tw_kd_in.d
adal_miss_char	This data array consists of characteristics and parameters describing an ADAT missile system and its performance constraints.	10	REAL	15 Hz	miss_atad.c	simnel/dala/ms_adchd
galat_miss_poly_deg	I list data array consists of values of the degree of each performed tequation used to compute the burn speed, the costs speed, maximum coolines of turns while unpowered, and coolines of turns while unpowered, and termoral his for the ADAT missile.	w	lui	15 Hz	miss_alad.c	See DESCRIFTION of Individual elements of TABLE
adal_bum_speed_coeff	This data array consists of the coefficients for a polynomial equation defining the ADAT missile burn speed with repect to time in the form using the Newton-Raphson method.	01	REAL	15 Hz	miss_atad.c	simnet/data/ms_ad_bs.d
adat_coast_speed_coeff	This data array consists of the coefficients for a polynomial equation defining the ADAT missile costs speed with repect to time in the	01	REAL	15 Hz	mlss_atad.c	simnet/data/ms_ad_cs.d
adal_bum_tum_coeff	form using the recover representation in the form using the polynomial equation defining the ADAT missile maximum cosine of turn while powered with represent to the representation of the form using the	10	REAL	15 Hz	miss_atad.c	simnel/data/ms_ad_bt.d
gasi_um_costl	The two to repute memors in the coefficients for a polynomial equation defining the ADAT missile maximum costen of turn while unpowered with repect to time in the form uning the Newton-Raphson method.	01	REAL	15 Hz	miss_alad.c	einnei/daia/ma_ad_ct.d

## TABLE 5.1. - SUMMARY of DATA ARRAYS [Continued]

NAME of DATA ARRAY	DESCRIPTION (NOTE 1)	SIZE of ARRAY	DATA TYPE (NOTE 2)	FREQUENCY of-	DECLARATION/DEFAULT MODULE	DATA SOURCE
adal_temp_bias_coeff	This data array condits of the coellidents for a polynomial equation defining the ADAT missile temporal bas with repect to time in the form using the Newton-Rapitson method.	10	REAL	15 Hz	miss_atad.c	simnet/data/ms_ad_tb.d
hellfr_miss_char	This data array concists of characteristics and parameters describing a Hellfire missile system and its performance constraints.	51	REAL	15 Hz	miss_hellfr.c	simnet/data/ms_hi_ch.d
hellfr_miss_poly_deg	This data array conditie of values of the degree of each polynomial equation used to compute the time-of-flight, the burn speed, and the cost speed for the Hellite missile.	c	JUJ	15112	miss_hellfr.c	See DESCRIPTION of individual elements of TABLE
hellfire_lof_coeff	This data array consists of the coefficients for a polynomial equation defining the Hellifre missile time-of-flight with repeat to time in the form using the Newton-Raphson method.	10	REAL	15 Hz	miss_hellfr.c	sinnet/data/ms_h_tt.d
helllire_bum_speed_coell	This data array consists of the coefficients for a polymontal equation defining the Hellitre missile burn speed with repect to range in the form using the Newton-Raphson method.	10	REAL	15112	miss_hellfr.c	simnet/data/ms_hf_bs.d
hellitre_coast_speed_coeff	This data array consists of the coefficients for a polynomial equation defining the Hellifre minssile coast speed with repect to time in the form using the Newton-Raphson method.	01	REAL	15112	miss_hellfr.c	simnet/data/ms_h_ca.d
kem_miss_char	This data array consists of characteristics and parameters describing a KEM missile system and its performance constraints.	01	REAL	15 Hz	misskem.c	sinnet/data/me_km_ch.d
kem_miss_poly_deg	This data array consists of values of the degree of each polynomial equation used to compute the burn speed, the costs apeed, maximum cosines of turns while powered and maximum cosines of turns while unpowered for the KEM missile.	S	int	15 Hz	misskem.c	See DESCRIPTION of Individual
kem_bum_speed_coell	This data array consists of the coefficients for a polynomial equation defining the KEM missile burn speed with repect to time in the form status, the Newton-Raphson method.	10	REAL	15 Hz	mdsekern.c	simnet/data/ms_km_bs.d
kem_coast_speed_coeff	This data array consists of the coefficients for a polymornal equation defining the KEM missile coast speed with repect to time in the form using the Newton-Raphson method.	10	REAL	15 Hz	missken.c	simnet/data/ms_km_c&d
kem_bum_tum_coell	This data array consists of the coefficients for a polynomial equation defining the KEM missile maximum cosine of turn while powered with repect to time in the form using the Newton-Rapiscon method.	01	REAL	15 Hz	misskem.c	sunnet/data/ms_km_bt.d
kem_coasi_turn_coeff	This data array consists of the coefficients for a polynomial equation defining the KEM massile maximum cosine of turn while unpowered with repect to time in the form using the Newton-Raphson method.	01	REAL	15 Hz	misekem.c	sinnel/dala/ms_km_ct.d
maverick_miss_char	This data array consists of characteristics and parameters describing a Maverick missile system and its performance constraints.	15	REAL	15 Hz	mlss_maverck.c	simnel/data/ms_mk_ch.d

## TABLE 5.1. • SUMMARY of DATA ARRAYS [Continued]

NAME of DATA ARRAY	DESCRIPTION (NOTE 1)	SIZE of ARRAY	DATA TYPE (NOTE 2)	FREQUENCY of CALCALATION	DECLARATION/DEFAULT MODULE	DATA SOURCE
maverick_miss_poly_deg	This data array consists of values of the degree of each polynomial equation used to compute the burn speed and the coast speed for the Maverick missile.	2	int	15 Hz	miss_maverck.c	See DESCRIPTION of individual elements of TABLE
maverick_bum_speed_coeff_	This data array consists of the coefficients for a polynomial equation defining the Maverick missile burn speed with rayect to time in the form using the Newton-Rapiscon method.	ĸ	REAL	15 Hz	mlss_maverck.c	simnet/data/me_mk_bs.d
maverick_coasl_spreed_coeff	This data array consists of the coefficients for a polynomial equation defining the Maverick ministic coasts speed with repect to time in the form refree the Nowiew-Raphson method.	ĸ	REAL	15 Hz	mdss_maverck.c	sinnet/dala/ms_mk_cs.d
nlos_miss_char	This data array consists of characteristics and parameters describing a NLOS missle system and its performance constraints.	20	REAL	15 Hz	miss-nios.c	simnet/data/nss_nl_ch.d
nlos_miss_poly_deg	This data array consists of values of the degree of each polynomial equation used to compute the time-of-flight, the burn speed, and the const speed for the NLOS missile.	r.	tu lut	15.13z	miss-nlos.c	See DESCRIPTION of individual elements of TABLE
llos_bank_mud_soln	This data array condets of the coellidents for a polynomial equation defining the NLOS miselle burn spoed with respect to time in the form region the Noveren Rankorn method	v.	REAL	15 Hz	miss-nlos.c	dunet/data/ma_nl_bs.d
ulos_cosst_speed_coeff	This data array consists of the coefficients for a polynomial equation defining the NLOS missile costs typed with report to time in the contract of Nowton Parhera method.	w	RBAL	15 Hz	miss-nios.c	simnes/data/ms_nl_cs.d
stinger_miss_char	This data array consists of characteristics and parameters describing a Stinger missile system and its medicamente constraints.	51	REAL	15 Hz	miss_stinger.c	simnet/data/ms_st_ch.d
stinger_miss_poly_deg	This data array consists of values of the degree of each polynomial equation used to compute the burn speed and the coast speed for the characterists.	2	ri.	15 Hz	miss_stinger.c	See DESCRPTION of individual elements of TABLE
stinger_bum_speed_coeff	Singer instance.  This data array consists of the coefficients for a polynomial equation defining the Singer insistle burn greed with report to time in the form using the Newton-Rabbson method.	2	REAL	15 112	miss_stinger.c	simnet/data/ms_st_bs.d
stinger_coasl_speed_coeff	This data array consists of the coefficients for a polynomial equation defining the Stinger missite coast speed with repect to time in the form using the Newton-Raphson method.	-	REAL	15 Hz	niss_sünger.c	sinnet/data/ms_st_cs.d
tow_miss_char	This data array consists of characteristics and parameters describing a TOW missile system and its performance constraints.	s	REAL	15 Hz	miss_tow.c	simnet/data/ms_tw_ch.d
tow_miss_poly_deg	This data array condats of values of the degree of a ach polymonial equation used to compute the burn speed, the coast speed, maximum coshine of turns while powered, and maximum coshine of turns while unpowered for the TOW missile.	vs	ţij	15 112	mles_tow.c	See DESCRIPTION of Individual elements of TABLE
low_burn_speed_coeff	This data array consists of the coefficients for a polymonial equation defining the TOW missile burn speed with repect to time in the form using the Newton-Raphson method.	ιn	REAL	15 Hz	mlss_tow.c	simuet/data/ms_tw_bs.d

## TABLE 5.1. • SUMMARY of DATA ARRAYS [Continued]

NAME of DATA ABBAY	DESCRIPTION	SIZE of	DATA TYPE	FREQUENCY of	DECLARATION/DEFAULT	DATA SOURCE
		ARRAY	MOIE 1	CALCULATION	mes tow.c	simmet/data/ms_tw_cs.d
low_coast_speed_coeff	This data array consists of the coefficients for a polymondal equation defining the TOW missile coast speed with repect to time in the form	s.	KEAL	Ē g		
low_bum_lum_coeff	using the Newton-Applicant interests of the coefficients for three polymorals equations is determined and a suray conditions is deveated, and downwards movement defining the TOW missile maximum cosine of turn white powered with prepect to time in the form using the Newton-	3×2	MAX_COS_COEFF	15 Hz	пізь_10м.с	simnet/dais/ms_tw_bt.d
tow_coast_tum_coeff	Rapiscon metroca.  This two-dimensional data array consists of the coefficients for three polymornial equations indewards, upwards, and downwards movement defining the TOW missile maximum costne of turn while unpowered with repect to time in the form using the	3×4	MAX_COS_COEFF	15 Hz	miss_low.c	sinnel/dala/ms_tw_ct.d
tow_miss_char (NOTE 3)	Newton-rapison interior. This data array consists of characteristics and parameters describing an ATGM missile	5	REAL	15 Hz	miss_algm.c	annet/data/mi_at_dr.d
low_miss_poly_deg (NOTE3)	system and use permission of the degree of each polynomial equation used to compute the burn opeced, the coast speed, in maximum cosines of turns while powered, and maximum cosines of turns while unpowered for the	s,	int	21151	miss_algn.c	See DES. AIT 1104 va nomes.
iow_burn_speed_coeff (NOTE 3)	ATCM mussile.  This data array consists of the coefficients for a polynomial equation defining the ATGM missile burn speed with repect to time in the	3	REAL	15 Hz	miss_atgm.c	simnet/data/ms_at_D&d
law_coast_speed_coefff (NOTE 3)	form using the Newton-Kapitson method. This data array consists of the coefficient for a polynomial equation defining the ATGM missule coast speed with repect to three in the	S	REAL	15 Hz	miss_aigm.c	simnet/data/ms_at_cs.d
tow_bum_tum_coeff (NOTE.3)	form using the Newton-Kephson method. This two-dimensional data array consists of the coefficients for three polynomial equations [sidewards, upwards, and downwards movement) defining the ATCM missile maximum cosine of turn while powered with respect to time in the form using the Newton-	3×2	MAX_COS_COEFF	IS Hz	miss_algm.c	simnei/data/ms_at_bt.d
tow_cosst_tum_coeff prore.3)	Raphson method.  This two-dimensional data array consists of the co-different for three polynomial equations [sidewards, upwards, and downwards movement] defining the ATGM muselle maximum cosine of turn while unpowered with report to time in the form using the	3×4	MAX_COS_COEFF	15 Hz	miss_algm.c	simnet/data/ms_at_ct.d
rki hydra char	Newton-Raphson method. This data array consists of chracteristics and	12	REAL	15 Hz	rki_hydra.c	simnet/data/rkt_hydr.d
hydra_rkt_char	parameters describing a Hyra V missie outer. This data array consists of characteristics and parameters describing a missile launcher evetem and its performance constraints.	7	REAL	15 Hz	rwa_hydra.c	simnet/data/rwa_hydr.d
-	System and the processing			į		

## TABLE 5.1. - SUMMARY of DATA ARRAYS [Continued]

					T HAT TO THE TANK THE	DAIA SOURCE
NAME OF DATA ABBAY	DESCRIPTION	SIZE of	DATA TYPE	FREQUENCY of	DECLARATION/DEFAULT	
LAME OF DATA AND A		ARRAY	[NOIE 1]	CALCULATION	TOO OH	cimpat /data/sub flec.d
2045 400 4	This data array consists of characteristics and	3	REAL	2H S1	sub_neco.c	
	narameters describing a Rechette flyout.				0.45	simpet/data/llec spd.d
	11. 1	5	KEAL	15112	מחם וופתור	
flechette_speed_coeff	I DIS CATA ACTION OF THE COLUMN OF THE COLUM					
	polynomial equation defining the nechette					
	flyout speed with repect to time in the form					
	neine the Newton-Raphson method.					dinnet/data/sub m73.d
	The state of the s	_	I REAL	15 Hz	3.C/M_0U4	
auh m73 char	This data array consists of characteristics and	,				
	-					

See individual TABLES for description of individual elements. 

NOTE 3

in it a "C" type for bitropic.
REAL is "C" musto OCIFAE for attracture of REAL types.

MAX\_COS\_COSIF is "C" musto DCIFAE for attracture of REAL types.

MAX\_COS\_COSIF is a "C" musto DCIFAE for attracture of REAL types.

The ALCA mixture models use the same data array names as the TOM mixture models. The Aurician names have been changed to reflect ATGAL. The modelse are used in separate builds.

## TABLE 5.1.1. - AERODYNAMICS DATA ARRAY

		1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	Г	DATA TVDE	BO TES SESTIMATES	CSU WHERE USED	DATA SOURCE
NAME OF DATA ELEMENT	DESCRETION	MEASURE	VALUE	[NOTE 1]	CALCULATED		
aero_data[ 0]	MOMENT_OF_INERTIA_X;	rad/sec	50000.0	REAL	default declaration rwa_aerodyn.c	Irwa_aerodyn.claerodyn_simul	simnet/data/rwa_aerod
mero_data[ 1]	MOMENT_OF_INERTIA_Y;	к8-ш•2	50000.0	REAL	default declaration rwa aerodyn.c	[rwa_aerodyn.c]aerodyn_init	simnet/data/rwa_aero.d
sero_data[ 2]	MOMENT_OF_INERTIA_Z;	kg-m•2	50000.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_int	simnet/data/rwa_aero.d
aero_data[ 3]	AIRFRAME_MASS;	kg	4881.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_sinul	simnel/data/rwa_acro.d
sero_data[ 4]	ORDINANCE_MASS;	kk	1591.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_simul	sinmet/data/rwa_acro.d
aero_data[ 5]	GRAV_CONSTANT;	m/sec**2	9.8	REAL	default declaration twa_aerodyn.c	[rwa_aerodyn.c]aerodyn_simut	simnet/data/rwa_aerod
sero_data[ 6]	.ככ־עכ־אי		0.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
sero_dats[7]	CG_AC_Y;		0.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[ B]	כפ־עכ־ב		-0.1	KEAL	delault declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_slinul	slinnet/data/rwn_acro.d
sero_data[ 9]	VIRTUAL_WING_AREA;	m.2	25.0	KEAL	default declaration rwa_aerodyn.c	(twa_aerodyn.c]aerodyn_simul	simnel/data/rwa_acro.d
aero_data[10]	VIRTUAL_WING_COP_AC_X;		0.0	KEAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c aerodyn_einul	slinnet/data/rwa_acro.d
sero_data[11]	VIRTUAL_WING_COP_AC_Y;		0:0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_Init	simnet/data/twa_aeo.d
aero_data[12]	VIRTUAL_WING_COP_AC_Z;		0.0	KEAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_init	simnet/data/rwa_aero.d
aero_data[13]	WING_LIFT_COEFFICIENT_FIT_3;		0:0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
sero_data[14]	WING_LIFT_COEFFICIENT_FIT_2;		0.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_data[15]	WING_LIFT_COEFFICIENT_FIT_T;		0.1	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
aero_dats[16]	WING_LIFT_COEFFICIENT_FIT_0;		0.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_simul	sinnet/data/rwa_aero.d
sero_data[17]	WING_STALL_AOA;	deg	30.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_eimul	simmel/data/rwa_acro.d
aero_data[18]	VSTAB_AREA;	m.2	R	REAL	default declaration rwa_aerodyn.c	Irwa_aerodyn.claerodyn_simul	simmet/data/rwa_acro.d
aero_data[19]	VSTAB_COP_AC_X;		0.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_simul	simmet/data/rwa_aero.d
sero_data[20]	VSTAB_COP_AC_Y;		-9.1	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_acro.d
aero_data[21]	VSTAB_COP_AC_Z;		0:0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_init	simnet/data/rwa_aerod
sero_data[22]	VSTAB_LIFT_COEFFICIENT_1;		2:0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.claerodyn_init	simnel/data/rwa_aero.d
aero_data[23]	VSTAB_STALL_SSA;	deg	0:09	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_acco.d
sero_data[24]	MAIN_ROTOR_COP_AC_X;		0.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_acro.d
aero_data[25]	MAIN_ROTOR_COP_AC_Y;		0.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aerod
aero_data[26]	NAIN_ROTOR_COP_AC_Z		2.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
sero_dsta[27]	MAIN_ROTOR_MAX_THRUST;	z	123500.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.claerodyn_init	[twa_aerodyn.c]aerodyn_simul	simmet/data/rwa_aecod

## TABLE 5.1.1. - AERODYNAMICS DATA ARRAY [Continued]

MAIN ROTOR MAST_TILL;         WAST_CONG         WAST_CONG         Inva. accopy, cheroly,	NAME OF DATA ELEMENT	DESCRIPTION	UNITS of	DEFAULT	DATA TYPE	CSU WHERE SET OR	CSU WHERE USED	DATA SOURCE
MANY MOTOR, MAX_FOLL, MONBNIT;   Nam   100000   REAL   default declaration range are objected by a first a seroly factor objected by a first a first a seroly factor objected by a first a first a seroly factor objected by a first a first a seroly factor objected by a first a first a seroly factor objected by a first a first a first a seroly factor objected by a first a first a seroly factor objected by a first a first a seroly factor objected by a first a f	ann date[28]	MAIN ROTOR MAST TILT:	MEASURE	VALUE 2.5	REAL	default dedaration rwa_aerodyn.c	[rwa_aerodyn.claerodyn_simul	simnet/data/rwa_aero.d
MANIN BOTOR, BANZ, LOAD, LONGUE         Nam         704760         REAAL         detail detailments a precipit.         If ma, a recolop, the recolop, that of the a precipit placed by the land of the analysis of the land of the analysis of the land of the a precipit placed by the land of the land	for intermed	THE TOWN TO LOW THE WILLIAM	c .			frwa aerodyn.claerodyn init		b crace curt atch toamic
MAIN, ROTOR, LANZ, PRICE, MONBENT;         Nam.         100000         REAL         detail detainments are support.         Inva., areolyncherolyn_time         4.           MAIN, ROTOR, LORGEL, COURTING.         Nam.         100000         REAL         detail detailments are support.         Inva., areolyncherolyn_time         1.           MAIN, ROTOR, LORGEL, COURTING.         0.0         REAL         detail detailments areolyn.         Inva., areolyncherolyn_time         1.           MAIN, ROTOR, COURTING.         0.0         REAL         detail detailments are areolyn.         Inva., areolyncherolyn_time         1.           MAIN, ROTOR, COURTING.         0.0         REAL         detail detailments are areolyn.         Inva., areolyncherolyn_time         1.           IAML, ROTOR, COV. AC. Y.         0.0         REAL         detail detailments are areolyn.         Inva., areolyncherolyn_time         1.           IAML, ROTOR, COV. AC. Y.         0.0         REAL         detail detailments are areolyn.         Inva., areolyncherolyn_time         1.           IAML, ROTOR, COV. AC. Y.         0.0         REAL         detail detailments are areolyn.         Inva., areolyncherolyn_time         1.           IAML, ROTOR, LORGEL, REAL         0.0         REAL         detail detailments are areolyn.         Inva., areolyncherolyn_time         Inva., areolyncherolyn_time	sero_data[29]	MAIN_ROTOR_MAX_LOAD_TORQUE;	u-X	76476.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.cjaerodyn_sunu	Mildlet/ data/ 1 Wa_act O.u
MANN_ROTOR_MONN_RESUL_MONNENT; N= n= 100000	sero_data[30]	MAIN_ROTOR_MAX_PITCH_MOMENT;	E	0.000001	KEAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_simul	simnel/data/rwa_aerod
MANN ROTOR_TORNUE_COUPLING.   0.5   REAL   default decision in a serolynic alreadyn_linid   1		MAIN_ROTOR_MAX_ROLL_MOMENT;	Ę.Ż	100000.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_init	simnet/data/rwa_aero.d
MANI, FORDR_CROUND_EFFECT_		MAIN_ROTOR_TORQUE_COUPLING_		0.5	REAL	default dedaration rwa_aerodyn.c [rwa_aerodyn.claerodyn_init.	[rwa_aerodyn.c]aerodyn_init	simnet/data/rwa_acro.d
1711_RD10R_COP_AC_X;   0.00   REAL   detail defaulton wa_aerolyn_cherolyn_inimal in the aerolyn_cherolyn_inimal intervention in the aero		MAIN ROTOR GROUND EFFECT		0.4	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
TANL_ROTOR_COP_AC_Y;		TAIL_ROTOR_COP_AC_X;		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.claerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aetod
1AIL_ROTOR_COF_AGZ_E   0.00   REAL   detail detail carboyine   1 two_ascodyn_cleredyn_simul   1 tol_ROTOR_COF_AGZ_E   N	sero_data[35]	TAIL_ROTOR_COP_AC_Y;		-9.1	REAL	default declaration rwa_aerodyn.c  rwa_aerodyn.claerodyn_init	[rwa_aerodyn.c aerodyn_simul	simnet/dala/rwa_aero.d
TAIL_ROTOR_MAX_LINUS; N	aero_data[36]	TAIL_ROTOR_COP_AC_Z		0.0	REAL	delault declaration rwa_aerodyn.c [rwa_aerodyn.claerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnel/data/rwa_aero.d
TAIL_ROTOR_MAX_LOAD_TORQUE; N°m 1684.8   REAL   default declaration rea_serolync   reva_serolync_lerectory laid   reva_serolync   reva_serolync_lerectory laid   reva_serolync_lerectory	sero_data[37]	TAIL_ROTOR_MAX_THRUST;	z	1.6068	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.claerodyn_init	[rwa_aerodyn.c]aerodyn_sinul	simnel/dala/rwa_aero.d
P_DRAC_COEFF_CONST;         00         REAL (default declaration raw a serodync lint)         (Ivva_areodync_berodyn_lint)           P_DRAC_COEFF_BREAK;         800         REAL (default declaration raw a serodync lint)         (Ivva_areodync_berodyn_lint)           P_DRAC_COEFF_BREAK;         0.00         REAL (default declaration raw a serodync lint)         (Ivva_areodync_berodyn_lint)           P_DRAC_COEFF_MAX;         0.00         REAL (default declaration raw a serodync lint)         (Ivva_areodync_berodyn_lint)           P_DRAC_COEFF_MAX;         0.00         REAL (default declaration raw_areodync_berodyn_lint)         (Ivva_areodync_berodyn_lint)           P_DRAC_COEFF_MAX;         0.00         REAL (default declaration raw_areodync_berodyn_lint)         (Ivva_areodync_berodyn_lint)           I DIAL_WEITED_SURFACE_AREA;         6.0         REAL (default declaration raw_areodync_berodyn_lint)         (Ivva_areodync_berodyn_lint)           I NAX_ATI_CIL_ANGLE_SIOI;         4s         REAL (default declaration raw_areodync_berodyn_lint)         (Ivva_areodync_berodyn_lint)           I NOVER_SLOYLINIT;         5.15         REAL (default declaration raw_areodync_berodyn_lint)         (Ivva_areodync_berodyn_lint)           I NOVER_AUG_PICTICAL_ROBLE_NORM;         4s         15.0         REAL (default declaration raw_areodync_berodyn_lint)           I NOVER_AUG_CINCLE_NORM;         4s         15.0         REAL (default declaration raw_areod	sero_data[38]	TAIL_ROTOR_MAX_LOAD_TORQUE;	ĘŻ	1684.8	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c}aerodyn_simul	simnet/data/rwa_aero.d
P_DRAC_IAS_BREAK         S80         REAL default declaration was aerodyn.         (Iva_aerodyn.claerodyn_init           P_DRAC_IAS_BREAK         602         REAL default declaration rwa_aerodyn.         (Iva_aerodyn.claerodyn_init           P_DRAC_IAS_MAX;         1000         REAL default declaration rwa_aerodyn.         (Iva_aerodyn.claerodyn_init           P_DRAC_IAS_MAX;         0.08         REAL default declaration rwa_aerodyn.         (Iva_aerodyn.claerodyn_init           P_DRAC_COEFF_MAX;         0.08         REAL default declaration rwa_aerodyn.         (Iva_aerodyn.claerodyn_init           P_DRAC_COEFF_MAX;         0.08         REAL default declaration rwa_aerodyn.         (Iva_aerodyn.claerodyn_init           P_DRAC_COEFF_MAX;         5.0         REAL default declaration rwa_aerodyn.         (Iva_aerodyn.claerodyn_init           NAX_AII_CIL_ANGLE_SIOP;         4.5         REAL default declaration rwa_aerodyn.         (Iva_aerodyn.claerodyn_init           IOVER_SLOW_LIMIT;         5.15         REAL default declaration rwa_aerodyn.         (Iva_aerodyn.claerodyn_init           INAX_AII_CIL_ANGLE_MORN;         deg         15.0         REAL default declaration rwa_aerodyn.         (Iva_aerodyn.claerodyn_init           INAX_AII_CIL_ANGLE_SIOW;         deg         15.0         REAL default declaration rwa_aerodyn.         (Ivw_aerodyn.claerodyn_init           INAX_AII_CIL_ANGLE_SIOW;	sero_data[39]	P_DRAG_COEFF_CONST;		0.0	REAL	default declaration rwa_serodyn.c [rwa_aerodyn.c]aerodyn_inti	[rwa_aerodyn.c]aerodyn_simul	simmet/data/rwa_acro.d
P_DRAG_COEFF_BREAK;	sero_data[40]	P_DRAG_TAS_BREAK		\$0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
P_DRAC_TAS_MAX;   100.0   REAL   default dedatation rea_aerodyn.claerodyn.lint   elima_aerodyn.claerodyn.lint   elima_aerodyn.claerodyn.claerodyn.lint   elima_aerodyn.claerodyn.lint	sero_data[41]	P_DRAG_COEFF_BREAK;		0.02	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_init	simnet/data/rwa_aerod
P_DRAG_COEFE_MAX;   0.06   REAL   default declaration rwa_aerodyn_clercodyn_simul   1.07AL_WETTED_SURFACE_AREA;   50.0   REAL   default declaration rwa_aerodyn_clercodyn_simul   1.07AL_WETTED_SUFFACE_AREA;   50.0   REAL   default declaration rwa_aerodyn_clercodyn_simul   1.00VER_SLOW_LIMIT;   1.00VER_SLOW_LIMIT;   1.00VER_AUG_FINED_SLOW_LIMIT;	sero_data[42]	P_DRAG_TAS_MAX;		100.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_init	simnet/data/rwa_aero.d
TUTAL_WETTED_SURFACE_AREA;	sero_data[43]	P_DRAG_COEFF_MAX;		90.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_init	[rwa_aerodyn.c]aerodyn_sûnul	simnet/data/rwa_acro.d
MAX_ATI_CIL_ANGLE_SIOP;         deg         6.0         REAL detail declaration rea_serodyn_clerodyn_list         Irwa_serodyn_clerodyn_list         Irwa_serodyn_clerodyn_list           NAX_ATI_DAMPING_FACTOR;         4.5         REAL detail declaration rea_serodyn_clerodyn_list         Irwa_serodyn_clerodyn_list         Irwa_serodyn_clerodyn_list           HOVER_SLOW_LIMIT;         5.15         REAL detail declaration rea_serodyn_clerodyn_list         Irwa_serodyn_clerodyn_list           MAX_ATI_CTL_ANGLE_NORM;         deg         15.0         REAL detail declaration rea_serodyn_clerodyn_list         Irwa_serodyn_clerodyn_list           MAX_ATI_CTL_ANGLE_NORM;         deg         15.0         REAL detail declaration rea_serodyn_clerodyn_list         Irwa_serodyn_clerodyn_list           MAX_ATI_CTL_ANGLE_SLOW;         deg         10.0         REAL detail declaration rea_serodyn_clerodyn_list         Irwa_serodyn_clerodyn_list           MAX_ATI_CTL_ANGLE_SLOW;         deg         10.0         REAL detail declaration rea_serodyn_clerodyn_list         Irwa_serodyn_clerodyn_list           INOVER_MED_LIMIT;         11.0VER_MED_LIMIT;         15.46         REAL detail declaration rea_serodyn_clerodyn_list         Irwa_serodyn_clerodyn_list           ATT_CTL_PITCTL_PITCTL_PITCTL_CANG         12.5         REAL detail declaration rea_serodyn_clerodyn_list         Irwa_serodyn_clerodyn_list           ATT_CTL_PITCTL_PITCTL_CANG         12.5	sero_data[44]	TOTAL_WETTED_SURFACE_AREA;		50.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.claerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnel/data/fwa_aero.d
MAX_ATT_DAMPING_FACTOR;	sero_data[45]	MAX_ATT_CIL_ANGLE_SIOP;	deg	6.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_simu	simnet/data/twa_aero.d
HOVER_ALOW_LIMIT;   S.15   REAL   default declaration rwa_aerodyn_cherodyn_lint	aero_data[46]	MAX_ATT_DAMPING_FACTOR;		4.5	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_einul	simnet/data/rwa_aero.d
HOVER_AUG_PITCH_RESET_VALUE;   0.004   REAL   default declaration rwa_serodyn.c arcolyn.c arco	aero_data[47]	HOVER_SLOW_LIMIT;		5.15	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
MIXX_ATT_CTL_ANGLE_NORM;   deg   15.0   REAL   default declaration rwa_serodyn.c   rwa_serodyn.cherodyn_stimul	sero_data[48]	HOVER_AUG_PITCH_RESET_VALUE;		0.044	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_sinul	simmet/data/rwa_aero.d
MAX_ATT_CTL_ANGLE_MED;   deg   10.0   REAL   default declaration rwa_serodyn_clercidyn_simul     Inva_serodyn_clercidyn_init   Irwa_serodyn_clercidyn_init     Irwa_serodyn_clercidyn_init   Irwa_serodyn_clercidyn_init     Irwa_serodyn_clercidyn_init   Irwa_serodyn_clercidyn_init     Irwa_serodyn_clercidyn_init   Irwa_serodyn_clercidyn_init     Irwa_serodyn_clercidyn_init   Irwa_serodyn_init     Irwa_serodyn_clercidyn_init   Irwa_serodyn_init     Irwa_serodyn_init   Irwa_serodyn_init     Irwa_serodyn_clercidyn_init   Irwa_serodyn_init     Irwa_serodyn_clercidyn_init   Irwa_serodyn_init     Irwa_serodyn_init   Irwa_serodyn_init     Irwa_serodyn_init   Irwa_serodyn_init     Irwa_serodyn_clercidyn_init   Irwa_serodyn_init     Irwa_serodyn_clercidyn_init   Irwa_serodyn_init     Irwa_serodyn_clercidyn_init   Irwa_serodyn_init     Irwa_serodyn_clercidyn_init   Irwa_serodyn_init     Irwa_serodyn_clercidyn_init   Irwa_serodyn_init     Irwa_serodyn_clercidyn_init   Irwa_serodyn_clercidyn_init     Irwa_serodyn_clercidyn_init   Irwa_serodyn_clercidyn_init     Irwa_serodyn_clercidyn_init   Irwa_serodyn_clercidyn_init     Irwa_serodyn_clercidyn_init   Irwa_serodyn_clercidyn_init     Irwa_serodyn_clercidyn_init   Irwa_serodyn_clercidyn_init     Irwa_serodyn_clercidyn_init   Irwa_serodyn_clercidyn_init   Irwa_serodyn_clercidyn_init     Irwa_serodyn_init   Irwa_sero	sero_dets[49]	MAX_ATT_CTL_ANGLE_NORM;	gap	15.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_eimul	simnel/dala/rwa_acro.d
MAX_AIT_CIL_ANGLE_SLOW;   deg   6.0   KEAL   default dedaration rwa_serodyn.claerody	sero_data[50]	MAX_ATT_CTL_ANGLE_MED;	gap.	10.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
HOVER_MED_LIMIT;   15.46   REAL   default declaration rwa_serodyn.clercodyn.clarcody	sero_dsta[51]	MAX_ATT_CTL_ANGLE_SLOW;	deg	0.9	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_init	sinnet/data/rwa_acro.d
ATT_CIL_PITCH_I_GAIN; 2.5 REAL default declaration rws_serodyn.cherodyn.cherodyn.cherodyn.eimul frws_serodyn.cherodyn.eimul frws_serodyn.cherodyn.eimul frws_serodyn.cherodyn.eimul	aero_data[52]	HÖVER_MED_LIMIT;		15.46	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_init	simmet/dats/rwa_aero.d
ATT_CIL_PITCH_I_GAIN; 0.05 REAL default dedaration rwa_serodyn.c [rwa_serodyn.c] [rwa_serodyn.	sero_deta[53]	ATT_CTL_PITCTI_P_CAIN;		2.5	REAL	delault declaration rwa_aerodyn.c [rwa_aerodyn.claerodyn_init	Irwa_aerodyn.cherodyn_simul	sinnel/data/rwa_aero.d
	sero_dats[54]	ATT_CTL_PITCH_I_GAIN;		0.05	REAL	default declaration rwa_aerodyn.c  rwa_aerodyn.claerodyn_init	[rwa_aerodyn.claerodyn_simul	simnel/data/rwa_aecod

## TABLE 5.1.1. - AERODYNAMICS DATA ARRAY [Continued]

	TOTAL CONTRACTOR	בעבועם -					
NAME OF DATA ELEMENT	DESCRIPTION	MEASURE	VALUE	[1016 1]	CALCULATED	ina serodon clastodon simul	sinunet/data/rwa_acro.d
sero_data[55]	ATT_CIL_ROLL_P_GAIN;		2:0	KEAL	default declaration rwa_aerodyn.c	I'ma_aerouyncherouyn_	7 - 7 - 7
	ATT CIL ROLL I GAIN:		90.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_simul	simnel/dala/rwa_aerou
sero_deta[36]	NIND ALL DON DE CAIN.		0.100	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.claerodyn_simul	elmmet/data/rwa_acro.d
sero_data[57]	HOVER AND CALLED		100	PEAI	default declaration twa aerodyn.c	[rwa_aerodyn.c]aerodyn_sinul	simnet/data/rwa_aecod
sero_data[58]	HOVER_AUG_ROLL_1_GAIN;		0.00	NEAL	[rwa_aerodyn.claerodyn_init	Irws serodyn claerodyn simul	sinnet/data/rwa_acrod
sero_data[59]	HOVER_AUG_PITCH_P_GAIN;		0.100	REAL	default declaration rwa_aerodyn.c	[I we_seriouym.barroum.	simpel/data/rwa aero.d
aero_data[60]	HOVER_AUG_PITCH_I_GAIN;		0.00	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_int	Irwa_aerodyn.Claerodyn_bintu	b One swal table to be
sero_data[61]	HOVER_AUG_YAW_P_GAIN;		10.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.claerodyn_init	[rwa_aerodyn.claerodyn_iiut	borne euriteich tormie
sero_data[62]	HOVER_AUG_YAW_I_GAIN;		5.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.claerodyn_int	bore ewr/esch/somis
sero data[53]	HOVER_AUG_CLIMB_P_GAIN;		1.0	KEAL	detault declaration rwa_aerodyn.c	rwa_aerodyn.cjaerodyn_sunu	one control (class)
sero data[64]	HOVER_AUG_CLIMB_LGAIN;		0.5	KEAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_eunu	binner/ data/ reach
	MAX STAB AUG PITCH ROLL CONTROL	-	0.20	KEAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_simit	man de la company de la compan
(coleran olas	NAX STAB AUG YAW CLINIB CONTROL	.;	90.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_simul	sinnet/data/rwa_aero.d
sero_datal bel	PATE DAMPING GAIN:		100000.0	REAL	default declaration rwa aerodyn.c	[rwa_aerodyn.c aerodyn_simul	simnet/data/rwa_aerod
sero_data[ b / j	MICHAEL DATE DAMPING CAIN:		100000.0	REAL	default declaration rwa aerodyn.c	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aerod
sero_data[68]	HICH-RAIE-DAWII ING. COM.		100000.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_acco.d
sero_data[69]	YAW_KAIE_DAMPHYG_CANN,		00000	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_sinul	simmet/data/rwa_aero.d
sero_data[70]	VERTICAL_RATE_DAMPING_CAIN;				frwa aerodyn.claerodyn init	[rwa_aerodyn.c]aerodyn_init	simmet/data/rwa_aero.d
aero_data[71]	LATERAL_VELOCITY_DAMPING_GAIN;		0.0001	KEAL	[rwa_nerodyn.c]aerodyn_init	frws serodyn.claerodyn init	sinnet/data/rwa_aerod
Act delef 721	LIFT COEFF VIRTUAL WING		9.0	REAL	default declaration rwa_aerodynuc frwa_aerodyn.cjaerodyn_init		property and from
72.30.000	OSWALD BEHC FACTOR:		6.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_einul	simnet/data/twa_seac.u
86ro_data[7.3]	INDITION DRAG COEFE	-	0.0	REAL	default dedaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_simul	simmet/dala/rwa_aeco.d
sero_data[/4]	CHAILMAN		0.0	REAL	default declaration rwa aerodyn.c	[rwa_aerodyn.c]aerodyn_simul	simmet/data/rwa_acco.d
asro_dats[75]	970 101		90	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_eimul	simnet/data/rwa_aero.d
sero_data[75]	NOT USED		3	BEAL	Irwa aerodyn.claerodyn intt default declaration rwa aerodyn.c	[rwa_aerodyn.c]aerodyn_slmul	simmet/data/rwa_aero.d
sero_data[77]	NOTUSED		3		Irwa aerodyn.claerodyn init	Irwa aerodyn.claerodyn_simul	simnet/data/rwa_aero.d
sero_data[78]	NOT USED		0.0	KE AL	[rwa_aerodyn.c]aerodyn_init	Irwa aerodyn.claerodyn_simul	simmet/data/rwa_aero.d
sero_dats[79]	NOT USED		0.0	KEAL	[rwa aerodyn.c]aerodyn Init	Irwa aerodyn.claerodyn simul	simmet/data/rwa_aero.d
sero_data[80]	NOT USED		0.0	REAL	(rwa aerodyn.claerodyn init	+	simmet/data/rwa_aero.d
11811	NOT USED		0.0	REAL	default degaration time, acrouping		

## TABLE 5.1.1. - AERODYNAMICS DATA ARRAY [Continued]

THOMAS OF THE PARTY	NOLLEGE	UNITS of	DEFAULT	DATA TYPE	CSU WHERE SET OR	CSU WHERE USED	DATA SOURCE
NAME OF DATA ELEMBAT		MEASURE	VALUE	[NOTE 1]	CALCULATED		b come sund that the contract
aero_dsta[82]	NOT USED		0.0	REAL	default declaration twa aerodyn.c	rwa_aerodyn.claerodyn_sunu	Situate / Unite / I wa and us
aero_data[83]	NOI USED		0:0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.claerodyn_simul	simnet/data/rwa_aerod
sero_data[84]	NOI USED		0.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_slinul	sinnet/data/rwa_aerod
sero_data[85]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.cherodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
sero_data[86]	NOT USED		0:0	KEAL	delault declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_sinul	slinnet/dala/rwa_acco.d
aero_data[87]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aerodyn_Init	[rwa_aerodyn.c]aerodyn_eimul	simust/data/rwa_aero.d
sero_data[88]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c aerodyn_simul	simnet/data/rwa_acro.d
aero_date[89]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
sero_data[90]	NOT USED		0.0	KEAL	default declaration twa_aerodyn.c	[rwa_aerodyn.c]aerodyn_sinul	simmet/data/rwa_aero.d
sero_dsta[91]	NOT USED		0.0	KEAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c aerodyn_simul	simnet/data/rwa_aero.d
sero_data[92]	NOI USED		0.0	REAL	delault declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_stinul	* simmet/data/rwa_acro.d
sero_data[93]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.claerodyn_simul	simnet/data/rwa_aero.d
aero_data[94]	NOT USED		0:0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c aerodyn_eimul	simmet/data/rwa_acro.d
sero_dats[95]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c frwa_aerodyn.claerodyn_init	[rwa_aerodyn.c]aerodyn_simul	simnel/data/rwa_aeco.d
sero_data[95]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_sinul	simnet/data/rwa_acco.d
sero_data[97]	NOT USED		0.0	REAL	delault declaration rwa_aerodyn.c	[rwa_aerodyn.c]aerodyn_simul	simnet/data/rwa_aero.d
sero_data[98]	NOT USED	-	0.0	REAL	default declaration rwa_aerodyn.c	[rwa_nerodyn.c]aerodyn_simul	similari / data/ twa_asa co
sero_data[99]	NOT USED		0.0	REAL	default declaration twa_aerodyn.c frwa_aerodyn.claerodyn_int	rwa_aerodyn.cjaerodyn_simui	billulei/ usis/ twa_st con

15 1 BEAL H. C. MAGGO DEFINE for type float.

# TABLE 5.1.2. - AERODYNAMICS INITIALIZATION DATA ARRAY

NAME of DATA	DESCRIPTION	UNITS of MEASURE	DEFAULT	DATA TYPE [1 TION]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
sero_init[ 0]	cyclic pitch		0.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aero_simul	sinmet/data/rw_ae_in.d
sero_init[ 1]	cyclic roll		0.0	REAL	default declaration rwa_aerodyn.c	[twa_aerodyn.c]aero_init	simnet/dala/rw_ae_in.d
aero_init[ 2]	collective		0.0	REAL	default dedaration rwa_aerodyn.c	[rwa_aerodyn.c]aero_init	slinnet/data/rw_ae_in.d
sero_init[ 3]	pedal		0.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aero_eimul	sinnel/data/rw_ae_in.d
sero_init[ 4]	stab aug pitch integrator		0.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.claero_simul	sinuet/data/rw_ae_in.d
sero_init[ 5]	stab aug roll integrator		0.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aero_slmul	simnet/data/rw_ae_in.d
aero_init[ 6]	stab aug yaw integrator		0.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.claero_simul	simnet/data/rw_ae_in.d
sero_init[ 7]	stab ang climb integrator		0.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.claero_simul	slmnet/data/rw_ae_in.d
sero_init[ 8]	attitude control pitch integrator		0.0	KEAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aero_stinul	simmet/data/rw_ae_in.d
sero_krit[9]	attitude control roll integrator		0.0	REAL	default declaration rwa_aerodyn.c	Irwa_aerodyn.claero_simul	sinnet/data/rw_ae_in.d
aero_init[10]	hover aug pitch integrator		0.0	REAL	delault declaration rwa_aerodyn.c	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_in.d
sero_Init[11]	hover aug roll integrator		0.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aero_sinul	simnet/data/rw_ae_in.d
aero_Init[12]	hover aug pitch angle		0:0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_in.d
aero_init[13]	hover aug roll angle		0.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_in.d
sero_init[14]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c		simnet/data/rw_ae_in.d
sero_init[15]	NOT USED	,	0.0	REAL	default declaration rwa_aerodyn.c		simnet/data/rw_ae_in.d
sero_Init[16]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c		simnet/data/rw_ae_in.d
aero_Irit[17]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c		simnet/data/rw_ae_in.d
sero_Init[18]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c (rwa_aerodyn.claero_init		simnel/data/rw_ae_in.d
sero_init[19]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c		siimet/data/fw_ae_in.d

NOTE 1 REAL H & "C" macro DEFNE for type flow.

## TABLE 5.1.3. - AERODYNAMICS SIMPLE DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS OF MEASURE	DEFAULT	DATA TYPE [NOIE 1]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
mero_simple[ 0]	MAX_HELICOPTER_POWER;	z	900000	REAL	delault declaration rwa_aerodyn.c	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_sp.d
aero_simple[ 1]	MAX_IIII;	pes	0.5	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aero_init	sinnet/dala/rw_ae_sp.d
aero_simple[ 2]	H.Kl; gain on position error		48.0	KEAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aero_inlt	sinnet/data/rw_ae_sp.d
sero_simple[ 3]	H_K2: gain on gravity term of power setting		0.15	KEAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_sp.d
sero_simple[ 4]	H_K7; air drag coellident		10.0	REAL	default declaration rwa_aerodyn.c	[twa_aerodyn.c]aero_sinul	sinnet/data/rw_ae_sp.d
aero_simple[ 5]	H_K8; air drag coellicient	k6	0.001	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_sp.d
aero_simple[ 6]	II_KP; power		150000.0	KEAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_sp.d
sero_simple[ 7]	H_KPR; pitch/roll constant, approximately pl/3		1.5	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.claero_simul	sinnet/data/rw_ae_sp.d
zero_simple[ B]	11_KY; yaw constant, approximately pl/2		0.7	KEAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aero_simul	sinnet/data/rw_ae_sp.d
sero_simple[ 9]	H_KH; hoverhold gain on velocity term		0.03	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c}aero_simul	sinnet/data/rw_ae_sp.d
sero_simple[10]	H_CHIE; collective hover hold gain		400000.0	REAL	delault declaration rwa_aerodyn.c	[rwa_aerodyn.c]aero_simul	sinnet/data/rw_ae_sp.d
sero_simple[11]	H_CL; coellicient of lift		0.001	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_sp.d
sero_simple[12]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c		simmet/data/rw_ae_sp.d
sero_simple[13]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_intt		simmet/data/rw_ae_sp.d
sero_simple[14]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c		simnet/data/rw_ae_sp.d
aero_simple[15]	NOT USED		0.0	KEAL	default declaration rwa_aerodyn.c		simnet/data/rw_ae_sp.d
sero_simple[16]	NOT USED		0:0	REAL	default declaration rwa_aerodyn.c		slinnet/data/rw_ae_sp.d
sero_simple[17]	NOT USED		0.0	KEAL	delault declaration rwa_aerodyn.c frwa_aerodyn.chero_intt		simnet/data/rw_ae_sp.d
aero_simple[18]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c  rwa_aerodyn.chero_int		sinnet/data/rw_ae_sp.d
aem_simple[19]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c		sinnet/data/rw_ae_sp.d

11 REAL 11 a "C" macro DEFINE for type flow.

## TABLE 5.1.4. - AERODYNAMICS STEALTH DATA ARRAY

ELEMENT  BEOG_Stealth( 0)   II_FN  BEOG_Stealth( 1)   II_SII  BEOG_Stealth( 2)   II_CC  BEOG_Stealth( 3)   MAX.  BEOG_Stealth( 4)   MAX.	II_EWD_MUL; H_SIDE_MUL;	MEASORE	80.0			land a second	cimpot/data/rw ae sl.d
	DE_MUL;		•	KEAL	default declaration rwa aerodyn.c	[rwa_aerodyn.cjaero_summ	
	DE_MUL;		0.00	REAL	default dedaration rwa aerodyn.c	[rwa_aerodyn.c]aero_stealth	simmet/data/rw_ae_sl.d
			2		Irwa aerodyn.claero stealth	1	of man / dala/ to all d
	H_COLL_MUL;		10.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_stealth	irwa_aerodyn.cjaero_steatut	מוויחובי ( משוש / משוש / משוש /
	MAX_TORQUE;		10000000000	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aero_simul	simmet/dala/rw_ae_sl.d
	MAX_PURCE;		10000000000	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aero_simul	simnet/data/rw_ae_sl.d
ages stealth[ 5] MASS;	\$5;	kg	5000.0	REAL	default dedaration rwa_aerodyn.c	[rwa_aerodyn.c]aero_eimul	simnet/data/rw_ae_st.d
	INERTIA;		25000.0	REAL	default declaration rwa_aerodyn.c	[rwa_aerodyn.c]aero_shnul	simnei/data/rw_ae_sl.d
sero_stesith[ 7] DEA	DEAD_ZONE;		0.03	REAL	default declaration rwa_serodyn.c [rwa_acrodyn.claero_stealth	[rwa_aerodyn.c]aero_simul	simnet/data/tw_ac_m.u
sero_stealth[ 8]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c		פווווווווווווווווווווווווווווווווווווו
aero_steaith[ 9]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c		sinnel/data/rw_ae_sl.d
aero stealth(10]	NOT USED		0.0	KEAL	default declaration rwa_aerodyn.c		sinuret/data/rw_ae_fd.d
aero_stealth[11]	NOT USED		0.0	REAL	default declaration twa aerodyn.c		simmet/data/rw_ae_st.d
aero stealth[12]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c	-	sinuret/data/rw_ae_st.d
aero stealth[13]	NOT USED		0.0	REAL	default declaration twa_aerodyn.c		simnet/dala/rw_ae_st.d
sero_stesith[14]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c		simnel/dala/rw_ae_sl.d
sero stealth[15]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c		simnel/dala/rw_ae_sl.d
aero_stealth[16]	NOT USED		0:0	REAL	default declaration rwa_aerodyn.c		simmet/data/rw_ae_st.d
aero_stealth[17]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c [rwa_aerodyn.c]aero_atealth		simnet/data/fw_ae_ss.d
aero_stealth[18]	NOT USED		0.0	REAL	default dedaration rwa_aerodyn.c [rwa_aerodyn.c]aero_stealth		simmel/dala/rw ae si.d
aero stealth[19]	NOT USED		0.0	REAL	default declaration rwa_aerodyn.c		

REAL IS a "C" made DEFINE for type float.

### TABLE 5.1.5. - ENGINE DATA ARRAY

NAME of DATA	DESCRIPTION	UNITS OF MEASURE	DEFAULT	DATA TYPE (NOTE 1)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
engine_date[ 0]	GOVERNOR_ENGINE_SPEED_SETTING;	rad/sec	1030.55	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	sinnet/data/rwa_engn.d
engine_data[ 1]	GOVERNOR_P_GAIN;		0.05	REAL	default declaration rwa_engine.c frwa_engine.clongine_init	[rwa_engine.c engine_init	simmet/data/rwa_engn.d
engine_data[ 2]	COVERNOR_I_GAIN;		0.05	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_Init	[rwa_engine.c engine_init	simmet/data/rwa_cngn.d
engine_data[ 3]	MAX_ENGINE_TORQUE;	Ę.Ż	1031.6	KEAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	sinnet/data/rwa_engn.d
engine_data[ 4]	NIN_ENGINE_LOAD_TORQUE;	E-Z	25.0	KEAL	default declaration rwa_engine.c frwa_engine.clengine_init	(rwa_engine.c]engine_simul	sinnet/data/rwa_engn.d
engine_data[ 5]	MAX_ENGINE_PERCENT_POWER;	percent	1.2	REAL	default declaration rwa_engine.c	[rwa_engine.c]engine_simul	sinnet/data/rwa_engn.d
engine_data[ 6]	ENGINE_TORQUE_INTERCEPT;		1200.0	REAL	default declaration rwa_engine.c [rwa_engine.clengine_init	[rwa_engine.c]engine_simut	sinnet/data/rwa_engn.d
engine_data[ 7]	ENGINE_TORQUE_SLOPE;		0.16438	KEAL	default declaration rwa_engine.c frwa_engine.clengine_init	[rwa_engine.c]engine_simul	sinnet/data/rwa_engn.d
engine_data[ B]	NOSE_GEARBOX_RATIO;		2.130	KEAL	default declaration rwa_engine.c [rwa_engine.clengine_init	[rwa_engine.c]engine_simul	sinnet/data/rwa_engn.d
engine_data[ 9]	NIAIN_ROTOR_GEAR_RATIO;		34.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	Irwa_engine.clengine_simul	sinmet/data/rwa_mgn.d
engine_data[10]	IAIL_KOTOR_GEAR_KATIO;		7.0	KEAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c engine_simul	sinnet/data/rwa_engn.d
engine_data[11]	POWERTRAIN_INERTIA;		0.001	REAL	delault declaration rwa_engine.c frwa_engine.clengine_init	(rwa_engine.clengine_simul	sinnet/data/rwa_engn.d
engine_data[12]	MAX_FUELFLOW;	gals/hr	153.8461539	KEAL	default declaration rwa_engine.c frwa_engine.cjengine_init	[rwa_engine.c]engine_simul	sinnet/data/rwa_cngn.d
engine_data[13]	NOT USED		0.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_data[14]	NOT USED		0.0	REAL	default declaration rwa_engine.c	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_data[15]	NOT USED		0.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	Irwa_engine.cjengine_simul	sinnet/data/rwa_engn.d
engine_data[16]	NOT USED		0.0	REAL	default declaration rwa_engine.c frwa_engine.c]engine_int	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_data[17]	NOT USED		0.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.clengine_simul	simnet/data/rwa_engn.d
engine_date[18]	NOI USED		0.0	KEAL	default declaration rwa_engine.c [rwa_engine_fult	frwa_engine.clengine_simul	sinnet/data/rwa_engn.d
engine_data[19]	NOT USED		0.0	REAL	default declaration rwa_engine.c frwa_engine.clengine_int	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d

NOTE 1 REAL H 8 "C" macro DEFRE for type float.

## TABLE 5.1.6. - ENGINE INITIALIZATION DATA ARRAY

NAME OF DATA ELEMENT	DESCRIPTION	UNITS OF MEASURE	DEFAULT	DATA TYPE	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
engine_init_data[ 0]	engine power		0.0	REAL.	default declaration rwa_engine.c	Irwa_engine.chugine_simul	sinnet/data/rwa_engn.d
engine_kvit_data( 1)	engine percent torque		0.0	REAL	default declaration rwa_engine.c	[rwa_engine.c]engine_init	simnet/data/rwa_engn.d
engine_init_data[ 2]	دينونين واعتصرا		0:0	REAL	default declaration rwa_engine.c	[rwa_engine.clengine_init	sinnet/data/rwa_engu.d
engine_init_dats[ 3]	integrator gain		0.0	REAL	default declaration rwa_engine.c [rwa_engine_clengine_init	[rwa_engine.c]engine_simul	sinnet/data/rwa_engn.d
engine_hit_dats[ 4]	last percent shalt speed		0.0	REAL	default declaration rwa_engine.c frwa_engine.clengine_init	rwa_engine.c engine_sinul	stinnel/data/rwa_engn.d
engine_init_data[ 5]	last percent torque		1.0	REAL	default declaration rwa_engine.c frwa_engine.clengine_init	[rwa_engine.c]engine_simul	simnet/data/rwa_engn.d
engine_init_dats[ 6]	hours of flight		0.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init	Irwa_engine.c engine_simul	simnet/data/rwa_engn.d
engine_init_data[ 7]	NOT USED		0.0	REAL	default declaration rwa_engine.c [rwa_engine.c]engine_init		simnet/data/rwa_engn.d
engine_init_data[ 8]	NOTUSED		0.0	REAL	delault declaration rwa_engine.c frwa_engine.ckngine_init		sunnet/data/twa_engn.d
engine_init_data[ 9]	NOTUSED		0:0	REAL	default declaration rwa_engine.c [rwa_engine.clengine_init		simhet/data/fwa_engn.d

NOTE 1 REAL IS "C" MINTO DEFINE for type float.

## TABLE 5.1.7. - ENGINE STATUS DATA ARRAY

NAME OF DATA ELEMENT	DESCRIPTION	UNITS OF	DEFAULT	DATA TYPE INOTE 1]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
engine_stat_data[ 0]	minutes of flight		0	ii.	default declaration rwa_engine.c	Irwa_engine.c engine_simul	simnet/data/rwa_engn.d
engine_stat_dats[ 1]	old minutes of Hight		0	HIII	default declaration rwa_engine.c  rwa_engine.c engine_init	[rwa_engine.ckngine_simul	simnet/data/rwa_engn.d
engine_stat_data[ 2]	engine status		-	ıuı	default declaration rwa_engine.c frwa_engine.cjengine_init	[rwa_engine.c]engine_sunul	simnet/data/rwa_engn.d
engine_stat_deta[ 3]	starting engine		_	ii.	default declaration rwa_engine.c frwa_engine.clengine_init	[rwa_engine.c]engine_sinul	simnet/data/rwa_engind
engine_stat_data[ 4]	number of engines		7	lui	default declaration rwa_engine.c [rwa_engine.c]engine_inft	Irwa_engine.c/engine_simul	simnet/data/rwa_engn.d
engine_stat_data[ 5]	engine is damaged		0	ā	default declaration rwa_engine.c [rwa_engine.c]engine_init	[rwa_engine.c/engine_simu	sinnet/data/rwa_engn.d
engine_stat_data[ 6]	transmission is damaged		0	ᆵ	default declaration rwa_engine.c frwa_engine.cjengine_init	[rwa_engine.c engine_simul	sinnet/data/rwa_engn.d
engine_stat_data[ 7]	NOTUSED		0	il.	default declaration rwa_engine.c frwa_engine.clengine_init		sunnet/data/rwa_engh.d
engine_stat_data[ 8]	NOT USED		٥	ini	default declaration rwa_engine.c [rwa_engine.c]engine_init		sunnel/data/rwa_engn.d
engine_stat_data[ 9]	NOT USED		۰	int	default declaration rwa_engine.c [rwa_engine.c]engine_init		simnel/data/rwa_engn.d

## TABLE 5.1.8. - KINEMATICS DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT	DATA TYPE [NOTE 1]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
Knemat_data[ 0]	GRAV_CONSTANT;	m/nec**2	9.810	REAL	default dedaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematica_ int	[rwa_kinemat.c]engine_simul	simnet/data/rwa_kine.d
idnemat_data[ 1]	SIN_AOA_LIMIT;	deg	0.642787610	REAL	default dedaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_ init	[rwa_kinemat.c]veh_spec_kinematics_ init	sinnet/data/rwa_kine.d
idnemat_data[ 2]	COS_AOA_LIMIT;	gəp	0.766044443	REAL	default dedaration rwa kinemat.c [rwa_kinemat.c]veh_spec_kinematics_ init	[rwa_kinemat.c]veh_spec_kinematlcs_ init	simnet/data/rwa_kine.d
Knemat_data[ 3]	SIN_YAW_LIMII;	8ap	0.642787610	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematica_ init	rwa_kinemat.c}engine_simul	simnet/data/rwa_kine.d
Kinemat_data[ 4]	COS_YAW_LIMIT;	deg	0.766044443	REAL	default declaration rwa_kinematic [rwa_kinemat.c]veh_spec_kinematics_ init	rwa_kinemat.c]engine_simul	simnet/data/rwa_kine.d
Mnemat_data[ 5]	DISPLAY_SPEED_LIMIT;		5.0	REAL	default dedaration rwa_kinemat.c [rwa_kinemat.c]weh_spec_kinematios_ init	[rwa_kinemaLc]engine_simul	simnet/data/rwa_kine.d
Mnemst_data[ 6]	NOT USED		0.0	REAL	default dedaration rwa_kinemat.c  rwa_kinemat.cjveh_spec_kinematios_  intl		simnet/data/rwa_kine.d
Mnemat_data[ 7]	NOT USED		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematica_ init		sinnel/data/rwa_kine.d
Mnemat_data[ 8]	NOT USED		0.0	REAL	default declaration twa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_ init		simnet/data/rwa_kine.d
kinemat_data[ 9]	NOT USED		0.0	REAL	default dedaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_ init		simnet/dala/rwa_kine.d
kinemat_data[10]	NOT USED		0.0	REAL	default dedaration rwa_kinemat.c  rwa_kinemat.cjveh_apec_kinematics_  nit		simnet/data/rwa_kine.d
kinemat_data[11]	NOT USED		0.0	REAL	default dedaration rwa_khemat.c [rwa_kinemat.c]veh_spec_kinematica_ init		sinnet/data/rwa_kine.d
kinemat_data[12]	NOT USED		0.0	REAL	default declaration rwa_kineinat.c {rwa_kinemat.c]veh_spec_kinematios_ intt		sinnel/data/rwa_kure.d
kinemat_data[13]	NOT USED		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematios_ init		sinutet/dala/rwa_kine.d
kinemat_data[14]	NOT USED		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematica_ [rit]		simnet/dala/rwa_kine.d
kinemat_data[15]	NOT USED		0.0	REAL	default dedaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematios_ Intt		simnet/data/rwa_kine.d
kinemat_data[16]	NOT USED		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_ intt		simnet/data/rwa_kine.d
kinemat_data[17]	NOT USED		0.0	REAL	default declaration rwa_kinemat.c (rwa_kinemat.c]veh_spec_kinemation_ init		eimnet/data/rwa_kine.d
kinemat_data[18]	NOT USED		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_ int		eifinet/data/f wa_kine.d

## TABLE 5.1.8. - KINEMATICS DATA ARRAY [Continued]

NAME of DATA	DESCRIPTION	UNITS of	DEFAULT	DEFAULT DATA TYPE	CSU WHERE SET OR	CSU WHERE USED	DATA SOURCE
ELEMENT		MEASURE	VALLIE	[HOTE 1]	CALCULATED		
inemat data[19]	NOI USED		0.0	REAL	default declaration rwa_kdnemat.c		simnet/data/rwa_kine.d
•					[rwa_kinematc]veh_spec_kinematics_		
			_		Indi		

NOTE 1 REAL H a "C" macro DEFINE for type float.

# TABLE 5.1.9. - KINEMATICS INITIALIZATION DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MFASURE	DEFAULT	DATA TYPE (MOTE 1)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
knemat_init_data[ 0]	positive unit velocity in X axis		0.0	REAL	delault declaration rwa kinemat.c [rwa_kinemat.c]veh_spec_kinematica_ init	[rwa_kinemat.c engine_simul	simnet/data/rw_ki_in.d
knemat_init_data[ 1]	positive unit velocity in Y axis		1.0	REAL	default declaration rwa_kinemat.c  frwa_kinemat.c veh_spec_kinematics_  init	[rwa_kinemat.c]veh_spec_kinematics_ init	simnet/data/rw_ki_in.d
kinemat_init_data[ 2]	positive unit velocity in Zaxis		0.0	REAL	default dedaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematica_ init	[rwa_kinematc]veh_spec_kinematics_ init	sinnet/data/rw_ki_in.d
kinemaCinit_deta[ 3]	negalive unit velocity in X axis		0.0	REAL	default declaration rwa_kinematc [rwa_kinematc]veh_spec_kinematica_ init	rwa_kinemat.c]engine_simul	simnet/data/rw_Ki_in.d
kinamat_init_data[ 4]	negative unit velocity in Y axis		-1.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematica_ init	[rwa_kinemat.cjengine_simul	sinnet/data/rw_ki_in.d
kinemat_init_data[ 5]	negative unit velocity in Zaxis		0:0	REAL	default dedaration rwa_kinemat.c (rwa_kinemat.c)veh_spec_kinematics_ init	[rwa_kinemat.c]engine_simul	simnet/data/rw_ki_in.d
kinemat_init_date[ 6]	sine angle of attack		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematica_ init		sinnet/data/rw_ki_in.d
kinemat_init_data[ 7]	cosine angle of attack		1.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_ init		simnet/dala/rw_kl_in.d
kinemat_init_data[ 8]	sine yaw		0.0	REAL	default declaration rwa_kinematc  rwa_kinematc]veh_spec_kinematica_   init		emnet/data/tw_kl_in.d
kinemat_krit_data[ 9]	cosine yaw		0.1	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_ init		simnet/data/TW_Ki_In.d
kinemat_init_data[10]	altitude		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_ init		sunnet/data/rw_ki_in.d
idnemat_Init_data[11]	body pitch		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_ init		builty and the builty and
kinemat_Init_data[12]	body pitch offset		0:0	REAL	delault declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_ init		
Idnemat_init_data[13]	velodiy pilch		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_ init		sunnet/ oats/ fw_K_in.u

TABLE 5.1.9. - KINEMATICS INITIALIZATION DATA ARRAY [Continued]

THE PARTY OF THE PARTY	T DESCRIPTION	UNITS of	DEFAULT	DATA TYPE	CSU WHERE SET OR	CSU WHERE USED	
NAME OF DATA ELEMENT		MEASURE	VALUE	(NOTE 1)	CALCULATED		simnet/data/rw kl in.d
kinemat_init_data[14]	roll		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_ init		
kinemat_init_data[15]	heading		0.0	REAL	default dedaration rwa_kinemat.c Irwa_kinemat.c)veh_spec_kinematics_ init	[rwa_kinemat.c]engine_simul	sinnet/data/rw_ki_in.d
kinemat_Init_data[16]	true airs)wed		0:0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_ Init		sinnet/data/rw_kl_in.d
kinemat_init_data[17]	indicated airspeed		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_ init		sunnet/data/tw_kt_itcu
kinemat_init_data[18]	"g" lace		1:0	REAL	default declaration rwa_kinemat.c [rwa_kinematc]veh_spec_kinematice_ init		simnet/data/rw_ki_in.d
kinemat_init_data[19]	vertical speed		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_apec_kinematica_ init		bunnet/ data/ tw_m_ii
kinemat_init_data[20]	gravity component in X axis		0.0	KEAL	default declaration twa_kinemate [twa_kinemate]veh_spec_kinematica_ init		summer/ date/ tweet
kinemat_init_deta[21]	gravity component in Y axis		0.0	REAL	default declaration rwa_kinematc [rwa_kinematc]veh_spec_kinematics_ init	-	sunnel/data/rw_Ki_ih.d
kinemat_init_data[22]	gravity component in Z axis		-1.0	REAL	default declaration rwa_kinematc [rwa_kinematc]veh_spec_kinematics_ init		simnel/dala/rw_kd_in.d
kinemat_init_data[23]	normal velocity component in X axis		0.0	REAL	default dedaration rwa_kinemat.c [rwa_kinemat.c]veh_spec_kinematics_ init		simnet/data/rw_ki_in.d
kinemat_init_data[24]	normal velocity component in Y axis		0:1	REAL	default declaration rwa_kinemat.c [rwa_kinematc]veh_spec_kinematics_ inft		simnet/data/rw_ki_in.d
kinemat_Init_data[25]	normal velocity component in Z axis		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinematc]veh_spec_kinematica_ intt		sinnel/dala/rw_ki_in.d
kinemat_init_data[26]	NOT USEU		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinematc]veh_spec_kinematics_ int		simnet/data/rw_ki_in.d
kinemat_init_data[27]	NOT USED		0.0	REAL	default declaration rwa_kinemat.c {rwa_kinematc veh_spec_kinematics_ init		simnet/data/rw_ki_in.d
kinemat_init_data[28]	NOT USED		0.0	REAL	default dedaration rwa_kinemat.c [rwa_kinematc]veh_spec_kinematics_ init		simnel/data/rw_Ki_in.d
Idnemat_inlt_data[29]	NOT USED		0.0	REAL	default declaration rwa_kinemat.c [rwa_kinemat.c]veh_epec_kinematics_ init		sunnet/ data/ rw_ki_iii.d

NOTE 1 REAL Ha "C" macro DEFINE for type float.

#### TABLE 5.1.10 - HELLFIRE MISSILE CHARACTERISTICS DATA ARRAY

	DESCRIPTION	UNITS of	DEFAULT	DATA TYPE	CSU WHERE SET OR	CSU WHERE USED	DATA SOURCE
		MEASURE [NOTE 1]	١		•		to the left and a feet of the of
helifr_miss_char[ 0]	HELLFIRE_ARM_TIME; hellitre missite arm	licks	20.0	KEAL	delault declaration miss_hellin.c	imiss_heilir.clengine_simul	פוווחופו/ משנש/ ווופ־וווי חרמ
hellf_miss_char[ 1]	HELLERE BURNOUT HAIF; time of powered flight for helifire misale in ticks [24	licks	36.0	REAL	default declaration miss_hellir.c [miss_hellir.c]missile_hellire_init	(miss_hellfr.c)missile_hellfire_lnit	simmet/data/ms_hl_dh.d
hellfr_miss_char[ 2]	IJELLFIRE MAX_FLIGHT_TIME; maximum flight time for the helilite missule assumed in Hck-1% 0 seconds	ticks	540.0	REAL	default dedaration miss_hellfr.c [miss_hellfr.c]missile_hellfre_init	[miss_hellfr.c]missile_hellfire_Indt	simnet/data/ms_hf_ch.d
hellfr_miss_char[ 3]	SPEED_0;		30.95953043	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_helifr.c]engine_simul	simnet/data/ms_ht_ch.d
helifr_miss_char[ 4]	THETA_0;		0.046542113	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_hellfr.c]engine_simul	sinuret/dala/ms_hi_ch.d
helfr_miss_char{ 5}	SIN_UNGUIDE; sine of the delta pitch angle		0.069756474	REAL	default dedaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	imiss_hellir.cjengine_simul	simnel/data/ms_nf_ch.d
hellfr_miss_char[ 6]	COS_UNCUIDE; cosine of the delta pilch angle [4.0 degrees] for an unguided helifire missile		0.997564050	REAL	default declaration miss_hellfr.c [nuiss_helffr.c]missile_hellffre_init	[miss_hellfr.c engine_simul	simnet/dala/ms_hi_Ch.d
helifr_miss_char[ 7]	SIN_CLIMB; sine of the delta pitch angle [3.5		0.001072424	REAL	default declaration miss hellfr.c Imiss hellfr.cjmissile hellfire int	[miss_hellfr.c]engine_stinut	eimnet/dala/ms_ht_ch.d
hellfr_miss_char[ 8]	CUS_CLIMB; cosine of the delta pitch angle [3.5 degrees] for a dimbing hellfire missile		0.999991708	KEAL	default dedaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_init	imiss_hellfr.clengine_simul	simnet/data/ms_ht_dh.d
hellfr_miss_char[ 9]	SIN_LOCK; sine of the lock cone angle [9.0 degrees] for a locked-on hellfire missile		0.156434465	REAL	default declaration miss, heliftre finiss, helift clinissile, helifire, init	miss_hellfr.cjengine_simul	simmet/data/ins_in_cn.d
helifr_miss_char[10]	CUS LUCK; cosine of the tock cone angle 19.0 degrees] for a locked-on hellfire missile		0.987688341	KEAL	default declaration mass_neutr.c [md:se_hellfr.cjinisstle_hellfire_inft	[mails original state   control	of the by the by the by
hellfr_miss_char[11]	CUS_TERM; cosine of the terminal pitch angle		0.241921896	REAL	default declaration miss hellir.c Imiss hellfr.chnissile hellftre inft	Imiss_neuir.cjengine_simu	billules / dails / dails / dails
helfr_miss_char[12]	COS_LOSE; cosine of the pitch angle [20.0 degrees] for a lose-of-lock-on hellfire missile		0.939692621	REAL	default declaration miss helifr.c [miss helifr.c]missile helifire init	miss_hellfr.cjengine_simul	simmet/data/ms_in_di.d
helfr_miss_char[13]			0.0	REAL	detault declaration miss, helitr.c [miss_hellfr.c]missife_hellfire_inft	mise_neliff.cjengure_simui	Day Mary (1917)
helifr_miss_char[14]	NOTUSED		0.0	REAL	default declaration miss helifir. [miss helifi.chnissile helifire init	liniss_hellfr.cjengine_simui	Minist, unta/ me m. Cre

NOIE 1 one tek is equal to one frame or 1/15th of a see NOIE 2 REAL is a "C" macro DEPAE for type float.

- 388 -

## TABLE 5.1.11. - HELLFIRE MISSILE POLYNOMIAL DEGREE DATA ARRAY

						ייייי שתונטנ ווכנון	DA A SOURCE
NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT	DATA	CSU WHERE SET OR CALCULATED	CSO MARKE COST	
				MOTE 13	10 - 11 - 11 - 12 - 13 - 14 - 15 - 15 - 15 - 15 - 15 - 15 - 15	inter hallte hallte hallte hallte hallte lult; simnet/data/ms_hl_d.d	simnet/data/ms_hf_tf.d
helifire_miss_poly_deg[ 0]	HELLFIRE_TOF_DEC; polynomial degree for hellfre missile time-of-flight coefficient data		default: 4 range: 0 to 9	<u>=</u>	detault declaration miss_neuit.c; [miss_helff.c]missile_hellfire_fult	default declaration miss neutric, finiss_hellfr.c.philssule_hellfre_calc_tof	
						twice ballir clinicalle ballire init; simnet/data/ins.hl_bs.d	simnet/data/ins_hl_bs.d
Author miss and deaf 11	HELLFIRE BURN SPEED DEC; polynondal		default: 3	ij	dejault declaration miss_neur.c,	[miss_hellfr.c]missile_hellfire_fire;	
	degree for hellfire missile burn speed		range, o 10 7			Inise hellfr.chnissile helllire fly	14 ca / 4 ca / ca e
	coefficient data array		3.17.17.5	15	default declaration miss hellfr.c;	miss hellfr.c]missile_hellfire_lnit;   sunnet/data/mis_tu_con	eminel/data/ms_iu_cs-u
helifire_miss_poly_deg[ 2]	HELLIFIRE_COAST_SPEED_ DEG;		range: 0 to 9	i	[miss_hellfr.c]missile_hellfire_init	[miss hellfr.chnissile_hellire_lly	
	Very medicione data array						
	speed coemident dam and						

NOTE 1 but is a "C" type for integer.

TABLE 5.1.12. - HELLFIRE MISSILE TIME-OF-FLIGHT COEFFICIENT DATA ARRAY

						CELL WATER LICED	DATA SOURCE
NAME OF DATA ELEMENT	DESCRIPTION	UNITS of MFASURE	DEFAULT	DATA TYPE INOIE 2)	CALCULATED		
		INOIL 1			Called and the state of the sta	ings hellir chatsile helllire init;	simnet/data/ms_M_tf.d
	Satties missile time-of-flight coefficient and	ticks	18.0	REAL	default declaration mass_neurit.	miss hellfr.clmissile hellfire calc tof	
helifire_tof_coeff( U)	The title in a manufacture of the second of				THE PROPERTY OF THE PARTY OF TH	Indea holle cludselle hellitre int;	shrutet/data/ms_ld_tf.d
	Lattice missile time-of-flight coefficient as	ticks/meter	3.14618166-2	REAL	default dedaration mass ment of	finiss hellfr.clinissile hellfire calc tof	
helifire_tof_coeff[ 1]	TICHING THE THEORY OF THE THE THEORY OF THE THE THEORY OF THE THEORY OF THE THEORY OF THE THEORY OF THE THE THE THE THEORY OF THE THEORY OF THE THEORY OF THE THEORY OF THE				Links heldischmissischen mice hellfr.c	miss hellr.clmissile_hellfire_init	simmet/data/ms_M_II.d
	hallita missile hme-of-Hight coefficient at	ticks/m,2	3.19212746-6	KEAL	rate balle chaiseile hellfire init	[miss hellfr.c]missile hellfire calc tof	
helifire_tof_coeff[ 2]					it is a description mice beller	Imiss hellfr.chmissile hellifre init;	simnet/data/ms_M_tl.d
	in the state of the conflictent 21	ticks/m*3	3.52604136-10	REAL	default negativation in the figure in the	Imise hellfr.clmissile hellfire calc tof	
helifira_tof_coeff[ 3]	helling mussile united in the comment				Imiss neint chinesure menine	Imise helift clinissile hellitre init;	sinmet/data/ms_hf_tf.d
	and the state of t	licks/m**4	-28469594e-14	REAL	default dedaration muss mental	Imbe hallfe climisalle hellfire calc tof	
halifue tof coeff[ 4]	hellfire missile time-of-lingut coemocint 44				miss helfr.chnissue helifre mit	This items the shalls hallitee init.	simpet/data/me ld ti-d
		200 m/ 17:15	00	REAL	default declaration miss_hellir.c	Inds include the bolides calculated by	
Latter out coaff! 51	helline missile time-of-Hight coellident as	11/2001	;		Inuss helter.chnissile hellftre init	miss hellif cinusale licitite the	dimensi/data/ms hi Hd
			١	DEAI	default dedaration miss hellfr.c	[mies hellir.c]missile neilite iiu.	The state of the s
	ballitze missile time-of-flight coefficient a6	ticks/m.e	0.0	VEVE	Imies hellfr,chnissile hellfire init	[miss hellfr.c]missile hellitre calc tot	7 7 7 7
helifire_tof_coeff[ 6]					Astruit declaration miss heliff.c	nuss hellfr.c]missile helllire init;	simmel/data/tits_iu_ii.u
	hallite missile time-of-ilight coefficient a7	ticks/m°7	0:0	KEAL	imiss helifr.chulssile hellfire int	[miss_hellfr.c]missile_hellfire_calc_tof	E 31 34 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
helifire_tof_coerr, / J					Astrony Asslaration miss hellfr.c	[miss_hellfr.c]missile_helifire_ituit;	Minnel/data/ins_iu_ii.u
		ticks/m*8	0.0	KEAL	treating challes hellfire init	imiss hellfr.c]missile helifire calc tof	
hellfire_tof_coeff[ B]	Delinite indesite dance of an original and a second				illuse illentituding mise helling	imiss hellfr.clmissile hellifre init.	simnet/data/ms_N_tt.d
	in its in smand theht coefficient as	ticks/m*9	0.0	REAL	default degaration mass menne	imiss hellfr.clmissile hellfire calc tof	
helifire_tof_coeff[ 9]	Delinie mussile dine of ingris con in				lune menter direction		

OTE 1 one tick is equal to one frame or 1/15th of a se

## TABLE 5.1.13. - HELLFIRE MISSILE BURN SPEED COEFFICIENT DATA ARRAY

NAME OF DATA ELEMENT	DESCRIPTION	UNITS of MEASURE [MOTE 1]	DEFAULT	DATA TYPE  WOIT 2]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
helifire_burn_speed_coeff[ 0]	hellire missile burn spood coellicient ag	meters	2.0014395e-2	KEAL	default dedaration miss_hellfr.c [miss_hellfr.chmissile_hellfire_init	(miss_hellr.c missile_hellire_init;  miss_hellr.c missile_hellfire_fire;  miss_hellfr.c missile_hellfire_fiy	sinuret/data/ins_hf_bs.d
hellfire_burn_speed_coeff[ 1]	hellire missile burn speed coefficient a	m/tick	6.7384206e-1	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missüe_hellfire_init	miss_hellfr.c missile_hellfire_init;  miss_hellfr.c missile_hellfire_lfre;  miss_hellfr.c missile_hellftre_fly	einmet/data/ms_hf_bs.d
helifire_burn_speed_coeff[ 2]	hellire missile burn speed coefficient #2	m/tick"2	9.8007701e-3	REAL	default declaration miss_hellfr.c [nuss_hellfr.c]missile_hellfre_int	[miss_hellfr.c]missile_hellfire_init; [miss_hellfr.c]missile_hellfire_flre; [miss_hellfr.c]missile_hellfire_fly_	simnet/date/ms_ht_bs.d
hellfire_burn_speed_coeff[ 3]	hellire missile burn speed coellicient a3	m/tick**3	-1.6782227 <del>c-</del> 4	REAL	default dedaration miss, hellfr.c [miss_hellfr.c]missile_hellfire_init	[miss_helltr.c missile_helllire_init; [miss_hellfr.c missile_hellfire_fire; [miss_hellfr.c missile_hellfire_fiy	simnet/data/ms_ht_lx.d
hellire_burn_speed_coeff[ 4]	hellire nussile burn speed coellicient at	m/tick**4	0.0	REAL	default dedaration miss_hellir.c {miss_hellir.c}missile_hellire_init	Indes_hellr.cjmisalle_hellfire_fnit; [mdss_hellfr.cjmissile_hellfire_ffre; [miss_hellfr.cjmissile_hellfire_fly	simnet/data/ms_ht_bs.d
hellfire_burn_speed_coeff[ 5]	· hellitre missile burn speed coeffident as	m/tick**5	0.0	REAL	default declaration miss_hellfr.c [miss_hellfr.cjmissBe_hellfre_init	miss_hellir.c missile_hellire_ini;   indss_hellir.c missile_hellire_lire;   imiss_hellir.c missile_hellire_liy	simmet/data/ms_ht_bs.d
hellfre_burn_speed_coeff[ 6]	hellire missie burn speed coefficient ag	m/tick**6	0.0	REAL	default dechration miss_hellfr.c [miss_hellfr.chnissile_hellfre_init	Indss_hellfr.cjmlssile_hellire_init;  Indss_hellfr.cjmlssile_hellire_ifre;  miss_hellfr.cjmlssile_hellitre_ify	sinmet/data/ms_hl_bs.d
hellfre_burn_speed_cosff[ 7]	hellitre missile burn speed coellident a7	m/tick••7	0.0	REAL	default declaration miss_hellfr.c [miss_hellfr.c]missile_hellfire_imt	[miss_hellir.cjmissile_hellire_ini; [miss_hellir.cjmissile_hellire_fire; [niss_hellir.cjmissile_hellire_fly	simnet/data/ms_ht_bs.d
helifire_burn_speed_coeff[ 8]	hellitre missile burn speed coefficient ag	m/tick**8	0:0	KEAL	default dedaration miss_hellfr.c {miss_hellfr.c}missile_hellfire_init	inss helir cinassie heliire int fraiss helir cinassie hellire fire Imss helir cinassie heliire fly	simnet/data/ms_ni_bs.d
helifire_burn_speed_coeff[ 9]	hellfire missile burn speed coeffident ag	m/tick**9	0.0	REAL	default dedaration miss_hellfr.c [mise_hellfr.c missile_hellfire_init	(miss_hellfr.c/missile_hellitre_init; (miss_hellfr.c/missile_hellitre_fire; (miss_hellfr.c/missile_hellitre_fiy	simnet/data/ms_hi_bs.d

NOTE 1 one tick is equal to one frame or 1/15th of a 1 NOTE 2 REAL is a "C" made DEFINE for type float.

- 390 -

## TABLE 5.1.14. - HELLFIRE MISSILE COAST SPEED COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of	DEFAULT	DATA	CSU WHERE SET OR	CSU WHERE USED	DATA SOURCE
		MEASURE	VALUE	776	CALCULATED		
		[MOTE 1]		(NOIC 2)	,		
helifice coast speed coeff[ 0]	hellire missile coast speed coefficient ag	inclers	4.2733447e+1	REAL	default declaration miss, hellir.c	(miss_hellir.chnissile_hellire_init;	sinutet/data/ms_hl_cs.d
	•				[miss_hellfr.c]missile_hellfire_init	Inuss hellfr.chaissile hellfire fly	
helifue coast speed coeff[ 1]	hellire missile coast speed coefficient as	m/tick	4.10136136-1	KEAL	default declaration miss hellir.c	[miss_hellir.c]missile_helllire_init;	simnet/data/ms_ht_cs.d
	•				[mlss_helifr.c]missile_helifire_init	Iniss helife chulasile hellifre fly	
halfire const speed coeff[ 2]	hellire missile coast speed coefficient as	m/1kk*2	2.6023601e-3	KEAL	default declaration nuiss_hellir.c	[miss_hellir.chuksile_hellire_indt;	simmet/data/ms_ht_cs.d
					Indea hellfr.chnissile hellifre init	miss_helifr.chnisslie_hellifre_fly	
belifice coast speed coaff 31	hellfire nussile coast speed coefficient as	m/tick*3	-8.4870117e-6	REAL	default declaration miss_helifr.c	[miss_helifr.c]missile_helllire_init;	slmnet/data/ms_hf_cs.d
	•				[miss_hellfr.clmissile_hellfire_init	Inits hellfr.chnissile helifire fly	
halifine coast sneed coaff[ 4]	helitire nussile coast speed coefficient at	m/tick**4	1.3322932e-\$	REAL	detault declaration miss_hellfr.c	[miss_hellir.c]missile_hellire_init;	sinmet/data/ms_ht_cs.d
	•	,			(miss_hellfr.chnissile_hellfire_Init	(miss_hellfr.c/missile_hellfire_fly	
halfire cost eneed meff 51	hellifre missile coast speed coefficient as	m/tick**5	-7.9542005e-12	REAL	default declaration miss hellfr.c	[miss_hellfr.c]massile_helifire_init;	simnet/data/ms_ht_cs.d
					Imiss hellir.chnissile hellfire init	[miss_hellfr.c]mlssile_hellfire_fly	
halfire cost speed costff 61	helifire missile coast sixed coefficient ag	m/tlck**6	0.0	REAL	default declaration miss hellfr.c	(miss_hellfr.c)missile_hellfire_init;	simnet/data/ms_hf_cs.d
	•				[miss_hellfr.c]missile_hellfire_init	(miss_hellfr.chnissile_hellfire_fly_	
halifire cost speed costil 71	hellfire missile coast speed coefficient az	m/tick*7	0.0	REAL	default declaration miss hellfr.c	[miss_hellfr.c]missile_hellfire_init;	simnet/data/ms_hf_cs.d
					Imiss hellfr.chnissile hellfire int	Imiss hellfr.chnissile hellfire fly	
halifire coast speed coafff R1	helltire missile coast streed coefficient as	m/tick*8	0.0	REAL	default dedaration miss hellfr.c	Indes_helfir.cjndselle_helllire_init;	simnet/data/ms_hl_cs.d
	,	-			[miss_hellfr.chnissile_hellfire_init	Indes hellfr.chrissile hellftre fly	
halifie coset eneed metil 91	hellitre nussile coast streed coefficient ao	m/tick**9	0.0	REAL	default declaration miss_hellfr.c	[miss_hellfr.c]missile_hellfire_init;	simnet/data/ms_hf_cs.d
					miss hellfr.chnissile hellfire init	[miss_hellfr.chnissile_hellfire_fly	

NOTE 1 one tick is equal to one frame or 1/15th of a second

- 391 -

## TABLE 5.1.15 - MAVERICK MISSILE CHARACTERISTICS DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS OF MEASURE	DEFAULT	DATA TYPE	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
maverick_miss_char[ 0]	MAVERICK_ARM_IIME; maverick missile arm time delay before firing in ticks [1.3 seconds]	ucks	20.0	KEAL	delault declaration miss_maverck.cg   fmiss_maverck.cg   fmiss_maverck.chmissile_maverick_   fmiss_maverick_   fmiss_mav	indss_maverck.cjmissile_maverick_ily	kinnet/data/ms_mk_ch.d
maverick_miss_char[ 1]	MAVERICK, BURNOUT, TIME; time of powered Hight for maverick missile in ticks 11.5 seconds	ticks	22.5	REAL	default declaration miss_maverck.cg [miss_maverck.c]missile_maverick_ init	indsa_maverck.cimissile_maverick_ily	sinnet/data/ms_mk_ch.d
maverick_miss_char[ 2]	MAVERICK MAX_FLIGHT_HME; maximum flight time for the maverick missile assumed in ticks [60.0 seconds]	ticks	0.006	REAL	default declaration miss_maverck.c; [miss_maverck.c;missile_maverick_ init	[miss_maverck.c]missile_maverlck_ Init	simnet/data/ms_mk_ch.d
mavarick_miss_char[ 3]	MAVERICK_LCCK_THRESHOLD); cosine squared of the lock threshold angle for the maverick missile [6.0 degrees]		0.989073800	REAL	default declaration miss_maverck.c; [miss_maverck.c;missile_maverick_ init	[miss_maverck.c missile_maverlck_ Inl; [miss_maverck.c missile_maverick_ detectibility	sinnet/data/ms_mk_ch.d
maverick_miss_char[ 4]	MAVERICK_HOLD_THRESHOLD; cosine squared of the hold threshold angle for the mayorick missile [100 degrees]		0.969846310	REAL	default declaration miss_maverck c; [miss_maverck.c missile_maverick_ init	Indss_maverck.cIndssile_maverick_ detectibility	sinnet/data/ms_mk_dt.d
meverick_miss_char[ 5]	SPEED_0;		28.3333333	REAL	default declaration miss_maverck.c; [miss_maverck.c;missile_maverick_ [nit	iniss_maverck_cinússile_maverick_lly	simnel/data/ins_mk_ch.d
mavadck_miss_char[ 6]	THETA_0,		0.046542113	KEAL	delault declaration miss_maverck.c; (miss_maverck.c)missile_maverick_ init	Imiss_maverck.clmissile_maverick_ily	simnet/data/ins_mk_ch.d
maverick_miss_char[ 7]	SIN_UNCUIDE; sine of level Hight (0.0 degrees pitch) for an unguided maverick miselle		0.0	REAL	default declaration miss_maverck.c; [miss_maverck.chulssile_maverlck_ ]nlt	Imiss_maverck.c]missile_maverick_fly	elmnet/data/ms_mk_ch.d
mevenck_miss_char[ 8]	COS_UNCUIDE; cosine of level Hight [0.0 degrees pltch] for an unguided maverick missle		1:0	REAL	default declaration miss_maverck.c; [miss_maverck.chnissile_maverick_ Init	[miss_maverck.c]missile_maverick_fly	simnet/data/ms_mk_dh.d
maverick_miss_char[ 9]	SIN_CLIMB, sine of the delta pitch angle [3.5 degrees] for a climbing maverick missile		0.004072424	REAL	default declaration miss_maverck c [miss_maverck.c)missile_maverick_ init	[miss_maverck.c]missile_maverick_lly	simnet/data/ms_mk_ch.d
maverick_miss_char[10]	COS CLIMB; cosine of the delta pitch angle [3.5 degrees] for a dimbing maverick missile		0.999991708	REAL	default declaration miss_maverck.c [miss_maverck.c]missile_maverick_ init	(miss_maverck.c)missile_maverick_ify	sinnel/dala/ms_mk_dhd
maverick_miss_char[11]	SIN_LOCK; sine of the lock cone angle [5.0 degrees] for a locked-on maverick missile		0.087155743	REAL	default declaration miss_maverck.c [miss_maverck.c]missile_maverick_ init	[miss_maverck.c]missile_maverick_lly	smnel/data/ms_mk_Cn.d
maverick_miss_char[12]	COS_LOCK; cosine of the lock cone angle [5.0 degrees] for a locked-on maverick missile		0.996194698	REAL	default declaration miss_maverck.c; {miss_maverck.c}missile_maverick_ init	[miss_maverck.c]missile_maverick_lly	simmet/data/ms_mk_ch.d
maverick_miss_char[13]	COS_TERN; cosine of the terminal angle [80.0] degrees for a locked-on maverick missile		0.173648178	REAL	default declaration miss_maverck.c; [miss_maverck.c]missile_maverick_ init	[miss_maverck.c]missile_maverick_ily	simnet/data/ms_mk_ch.d
maverick_miss_char[14]	COS_LOSE; costine of the angle [20.0 degrees] for a loss-of-tock-on maverick missile		0.939692621	REAL	default declaration miss_maverck.c; [miss_maverck.c]missile_mavertck_   miss_maverck.c]missile_mavertck_	[miss_maverck_c missile_mavenck_ily	simnet/data/ms_mk_cn.d

- 392 -

## TABLE 5.1.16. - MAVERICK MISSILE POLYNOMIAL DEGREE DATA ARRAY

MAVERICK BURN SPED DEC.			TYPE	CALCULATED		
	SPIED DEC	default: 1	lut.	default declaration miss_maverck.c	default declaration miss_maverck.c; [miss_maverck.c]missile_maverick slinnet/data/ms_mk_bs.d	simmet/data/ms_mk_bs.c
polynomial degree for mare	polynomial degree for maverick missile burn speed coefficient data array	range: 0 to 9		[miss_maverck.c]missile_maverick	ints, maverck clinissile, maverick,	
	•			-	Imiss_maverck.chulastie_maverick_fly	
Ī	AAURONA COACT COCCO DEC.	default: 3	int	default declaration miss_maverck.c	default declaration miss maverck. [miss_maverck.c]missle_maverick_ simnet/data/ms_mk_cs.d	sinnel/dala/ms_mk_cs.
maverick_miss_poly_deg[ 1] MAVENICK_CASI_31 EE polynomial degree for mav	polynomial degree for maverick missile coast	range: 0 to 9		[miss_maverck.chnissile_maverick_	Inits, maverck clinisaile, maverick, Hy	

NOTE 1 Int Is a "C" type for integer.

## TABLE 5.1.17. - MAVERICK MISSILE BURN SPEED COEFFICIENT DATA ARRAY

			DEFAIR T		CCII WHERE SET OR	CSU WHERE USED	DATA SOURCE
NAME of DATA ELEMENT	DESCRIPTION	MEASURE	VALUE	TYPE	CALCULATED		
10 11111	mayorlek missile burn speed coefficient ap	$\perp$	0.0033333	KEAL	default declaration nuise maverck.c	[indes_maverck_christie_maverick_ sinuset/data/ins_mk_bs.d	sinuret/data/ms_mk_bs.d
may enck_burn_space_cooper	default is 67.0 m/soc				[miss_maverck.cpmissue_mavenck_	[miss maverck.c]missile_maverick_	
						lire;	
						Indes maverck.clmtssile maverick fly	
	manarich missile hurn streed coefficient 41;	m/tick*2	1.2577777	KEAL	default declaration miss_maverck.c	miss maverck chaissile maverick	sinmet/data/ms_mk_bs.d
mayerick_bum_speed_coeff( 1)	default is 274.9732662 m/sec "2				[miss_maverck.cjmissile_maverick	miss maverck.chnissile_maverick_	
						fire;	
						miss maverck curessie maverior ity	1. 24 Jun 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
mendet him spend coeff 2]	maverick nussile burn speed coefficient a2	m/lidx*3	0.0	REAL	default dedaration miss_maverck.c	[miss_maverck.c]missile_maverick_   sinner/ data/ nis_nik_0s.u	פוונתופו/ תפוש/ וווף דווא הפים
					Int	[miss_maverck.c]missile_maverick_	
-						tue; [miss mayerd:chrissile mayerick_fly	
		4	0.0	REAL	default declaration miss maverck.c	maverck.clmissile_maverick simnet/data/ms_mk_bs.d	simnet/data/ms_mk_bs.d
mayerick burn_speed_coeff[ 3]	maverick missile burn speed coefficient as	M/III/	3		Imiss maverck.chnissile_maverick_	Inte	
					trit	(miss_maverck.cjmissile_maverick_	
						[miss_maverck.chmissile_maverick_fly	
	and the state of t	m/lick**5	0.0	REAL	default declaration miss_maverck.c	[miss_maverck.c]missile_maverick_	simmet/data/ms_mk_bs.d
mayerick_bum_speed_coeff[ 4]	maverick intestite built speed comments				[miss_maverck.chmissile_maverick_	Int; [miss_maverck.c]missile_maverick_	
						fines maverck.chnissile_maverick_fly	

NOTE 1 one lick is equal to one frame or 1/15th of a succession of a succession of the forty of type float.

# TABLE 5.1.18. - MAVERICK MISSILE COAST SPEED COEFFICIENT DATA ARRAY

NAME OF DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT	DATA TYPE	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
maverick_coast_speed_coeff[ 0]	maverick missile coast speed coefficient age default is 327.2858074 m/sec		30.46972849	REAL	detault dedaration miss_maverck.c imiss_maverck.cimissile_maverick_ init	miss_maverck.c missile_maverick_   sinnet/data/ms_mk_cs.d init;   maverck.c missile_maverick_fly	sinnet/data/ms_mk_cs.d
maverick_coast_speed_coeff[ 1]	maverick missile coast speed coefficient a <sub>1</sub> ; m/lick**2 -9,7721 160k-2 default is -21,4609514 m/sec**2	m/tick*2	-9.7721160e-2	REAL	default declaration miss_maverck.c [miss_maverck.chnisstle_maverick_ init	default declaration miss_maverck.c   miss_maverck.chmissile_maverick_ simnet/data/ms_mk_csd   miss_maverdx.chmissile_maverick_chmissile_maverdx.fly     miss_maverdx.chmissile_maverdx.fly	simnet/data/ms_mk_cs.d
maverick_coast_speed_coeff[ 2]	maverick missile coast spreed coeliident 22 m/lick**3 1.2433925e-4 default is 0.8227650 m/sec**3	m/tick**3	1.2433925e-4	REAL	default dedaration miss_maverck.c [iniss_maverck.chnissile_maverick_ init	[miss_maverck.c missile_mavericksimnet/data/ms_mk_cs.d inly; [miss_maverck.c missile_maverick.fly]	simnet/data/ms_mk_cs.d
maverick_coast_speed_coeff[ 3]	maverick missile coast speed coellident ag m/lick"4 -54061501e-8 default is -0.0133200 m/sec"4	m/tick**4	-5.4061501e-8	REAL	default declaration miss_maverck.c [miss_maverck.chnissile_maverick_ init	[miss_maverck.c]missile_mavericksimnet/data/ms_mk_Gs.d init; [miss_maverck.c]andssile_maverick_fly	simnet/data/ms_mk_0s.0
maverick_coast_speed_coeff[ 4]	naverick missile coast speed coellicient a	m/lick**5	0:0	REAL	default dedaration miss_maverck.c  miss_maverck.cpnissile_maverlck_  ntl	[miss_maverck.c]missile_maverick_ Init: [miss_maverck.c]missile_maverick_fly	simnet/data/ms_mk_G.d

1 1 cone tick is equal to one frame or 1/15th of a seco

#### TABLE 5.1.19 - STINGER MISSILE CHARACTERISTICS DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT VALUE	DATA TYPE	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
stinger_miss_char[ 0]	SIINCER_BURNOUT_IIME; time of powered flight for stinger missile in ticks 1775 seconds!	licks	19.125	KEAL	detault dedaration mas_stinger.c; [miss_stinger.c]missile_stinger_ inft	iniss_stinger.cimissile_stinger_ init; [miss_stinger.cimissile_stinger_fly	simnet/data/ms_st_ch.d
stinger_miss_char{ 1}	STINGER_MAX_FLRCHT_TIME; maximum Stilght time for the stinger rulselle assumed in ticke 105 653 seconds.	licks	400.000	REAL	default dedaration miss_stinger.c; [miss_stinger.c]misslle_stinger_ init	Imise_stinger.cimisetie_stinger_ Init	sinmet/data/ms_st_du.d
stinger_miss_char[ 2]	STINGER, LUCK, HRESHOLD, cosine squared of the lock threshold angle for the stinger missile [125 degrees]		0.953153895	REAL	default declaration miss stinger c; [miss_etinger.c]missile_stinger_ init	imiss_stinger.c]mdssile_stinger_ inly: [miss_stinger.c]missile_stinger_ pre_launch: [mdss_stinger.c]missile_stinger_fly	eimnet/data/ms_st_ch.d
stinger_miss_char[ 3]	SPEED_0; default is 800.0 in/sec	m/tick	53,3333333	REAL	default dedaration miss_stinger.c fmiss_atinger.chrissile_stinger_ init	[miss_stinger.c]mdssile_stinger_ init; [miss_stinger.c]missile_stinger_fly	simnet/data/ms_st_ch.d
stinger_miss_char[ 4]	THETA_0, default is 15.0 deg/sec	rad/lick	0.0174	REAL	default dedaration miss_stinger.c; [miss_stinger.c}missile_stinger_ init	[miss_stinger.c]missile_stinger_ init; [miss_stinger.c]missile_stinger_fly	sinuici / data/ ms_st_cn.d
stinger_miss_char[ 5]	INVEST_DIST_SQ; detault distance is 300 m	E.2	90000.0	REAL	default declaration miss_stinger.c; [miss_stinger.c]missile_stinger_ init	[miss_stinger.c]ruseile_stinger_ Int; [miss_stinger.c]missile_stinger_fly	simnet/data/ins_st_ch.d
stinger_miss_char[ 6]	FUZE_DIST_SQ: detault distance is 20 m	II.22	400.0	REAL	default declaration miss_stinger.c; fmiss_stinger.cpnissile_stinger_ init	[miss_stinger.c missile_stinger_ init; [miss_stinger.c missile_stinger_fly	simnet/data/ms_st_ch.d
stinger_miss_char[ 7]	NOT USED		0.0	REAL	default declaration miss_stinger.c; {miss_stinger.c}missile_stinger_ init		slinnet/data/ms_st_dh.d
stinger_miss_char[8]	NOT USED		0.0	REAL	default declaration miss_stinger.c [miss_stinger.chnissile_stinger_ init		simmet/data/ms_st_ch.d
stinger_miss_char( 9)	NOT USED		0.0	REAL	default declaration miss_stinger.c; [miss_stinger.c]missile_stinger_ init		simnet/data/ms_st_ch.d
stinger_miss_char[10]	NOT USED		0.0	REAL	default declaration miss_stinger.c; [miss_stinger.c]missile_stinger_ init		sinunel/data/nis_st_ch.d
stinger_miss_char[11]	NOT USED		0.0	REAL	default declaration miss_sunger.c; [miss_stinger.c]missule_stinger_ init		simnet/data/ms_st_ch.d
stinger_miss_char[12]	NOT USED		0:0	REAL	default declaration miss_stinger.c; [miss_stinger.c]missile_stinger_ init		simnet/data/ms_st_ch.d
stinger_miss_char[13]	NOT USED		0:0	REAL	default dedaration miss_stinger.c [miss_stinger.c]missile_stinger_ Init		simnet/data/ms_st_ch.d
stinger_miss_char[14]	NOT USED		0.0	REAL	default declaration miss_stinger.cy [miss_stinger.c]missile_stinger_ init		simnet/data/ms_st_ch.d

one lick is equal to one frame or 1/15th of a secon

- 395 -

## TABLE 5.1.20. - STINGER MISSILE POLYNOMIAL DEGREE DATA ARRAY

						Carrie and the carrie	SATA COLIDOR
NAME OF DATA ELEMENT	DESCRIPTION	UNITS OF MEASURE	DEFAULT	DATA	CSU WHERE SET OR CALCULATED	CSU WHERE USED	TOWNS CIVI
						India attrace diministia etimost init	changet/data/ms at bad
the pale deef of	endemonial degree for stinger missile burn		_	Ξ	sunger.c	Intra Strate Change of the County of the Cou	
	0		_		Imiss stinger, chrissile stinger init		
	speed coefficient data array					in the other chalestle others init	change / data / ms at cs.d
11. 40.0	topmonial decree for ethnor missile coast		·	=	default declaration miss_sunger.c	Times bruiger, Chiusene suring a suring	
snuger_miss_poy_cell ! ]	١.				finise atinger clinissile atinger init		
_	Terror Carll Conflict arta						

NOTE 1 in it is a "C" type for mieger.

## TABLE 5.1.21. - STINGER MISSILE BURN SPEED COEFFICIENT DATA ARRAY

NAME OF DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT	DATA	CSU WHERE SET OR CALCULATED	CSJ WHERE USED	DATA SOURCE
stinger_burn_speed_coeff[ 0 ]	stinger_burn_speed_coeff[ 0] stinger missile burn speed coefficient #0	m/tick	6:1	REAL	default dedaration miss_stinger.c [miss_stinger.c]missile_stinger_init	[miss_stinger.c]missile_stinger_init; [miss_stinger.c]missile_stinger_fire; [miss_stineer.c]missile_stinger_fiy	simnet/data/ma_st_bs.d
stinger_burn_speed_coeff[1] stinger missile burn speed	coefficient at	ın/tick*2	in/iick**2 2689324619 REAL	REAL	default dedaration miss_stinger.c	imiss_stinger.cimissile_stinger_init; [miss_stinger.cimissile_stinger_fire;	simmet/data/ms_st_ba.d
						miss stinger.cmissile slinger ily	

NOTE 1 one tick to equal to one frame or 1/15th of a second NOTE 2 REAL is a "C" made DEFINE for type float.

# TABLE 5.1.22. - STINGER MISSILE COAST SPEED COEFFICIENT DATA ARRAY

NAME OF DATA ELEMENT	DESCRIPTION	UNITS OF MEASURE	DEFAULT	DATA	CSU WHERE SET OR CALCULATED	CSJ WHERE USED	DATA SOURCE
stinger_coast_speed_coeff[ 0]	Hinger missile coast speed coellident ag	m/tick	56.73662833	REAL	default dedaration miss_stinger.c [miss_stinger.c]missile_stinger_init	miss_stinger.c]missile_stinger_init; {miss_stinger.c]missile_stinger_itre; [miss_stinger.c]missile_stinger_itv	simnet/data/ms_st_cs.d
stinger_coast_speed_coeff[1]	stinger missile coast speed coefficient as	m/iick*2	-0.182369351	REAL	default declaration miss_stinger.c [miss_stinger.c missile_stinger_init	[miss_etinger.c]missile_stinger_init; [miss_stinger.c]missile_stinger_fire; fmiss_stinger.c]missile_stinger_fiy	sinmet/data/ms_st_ts.d
stinger_cosst_speed_coeff[ 2]	stinger missile coast speed coefficient a2	m/tlck*3	m/tick**3 23302001e-4	REAL	default declaration miss_stinger.c [miss_stinger.c]missile_stinger_init		simnet/data/ms_st_ca.d
stinger_coast_speed_coeff[ 3]	stinger missile coast speed coefficient ag	m/tick**4	m/lick**4 -1.0176282e-7	REAL	default dedaration miss_sitnger.c	[miss_stinger.c]missile_stinger_int; [miss_stinger.c]missile_stinger_fire; [miss_stinger.c]missile_stinger_fly	simnet/data/ms_st_cs.d

NOTE 1 one tick in equal to one frame or 1/15th of a secon NOTE 2 REAL is a "C" macro DEFRE for type float.

#### TABLE 5.1.23 - TOW MISSILE CHARACTERISTICS DATA ARRAY

Titalia or and a second	NOTIFICATION	UNITS of	DEFAULT	DATA	CSU WHERE SET OR	CSU WHERE USED	DAIA SOURCE
NAME OF DATA ELEMENT		MEASURE	VALLUE	TYPE Note 21	CALCINATED		
tow_miss_char[ 0]	TOW BURNOUL TIME; time of powered	licks	24.0	KEAL	default declaration miss_tow.c; [miss_tow.c]missile_tow_init	[miss_tow.c]misslle_tow_init; [miss_tow.c]misslle_tow_fly	simnet/dala/ms_tw_dh.d
tow_miss_cher[ 1]	TOW_RANGE_LIMIT_IIME; range limit time for the fow missile in ticks 17.89 seconds.; at the range in the state of the fow missile in the state of the massile is	icks	268.35	REAL	default declaration miss_tow.c; [miss_tow.c]missile_tow_init	Imiss_tow.cjmissile_tow_init; [miss_tow.cjmissile_tow_fly	simnet/data/ms_lw_ch.d
tow_miss_char[ 2]	allowed to fly to the maximum flight time TOW MAX FLIGHT TIME; maximum flight time for the tow missile in ticks; cosine of the	ticks	300.00	REAL	default declaration miss_tow.cg [miss_tow.cjmissile_tow_init	[miss_tow.c]missile_tow_liut	simnet/data/ms_tw_ch.d
	max turn is greater than 1.0 layond this point		6	1450	default declaration miss tow.c.	miss tow.c)missile tow init	sinmet/data/ms_tw_ch.d
tow_miss_char( 3)	NOT USED		25	News	[miss_tow.c]missile_tow_init	1301	clumot/data/me tw chd
tow_miss_char[ 4]	NOT USED		0.0	REAL	default declaration miss_tow.c; [miss tow.c]missile_tow_init	[miss_tow.c]missite_tow_iiut	7 ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) ( ) (

NOTE 1 One likk is equal to me frame or 1/15th of a second NOTE 2 REAL is a "C" made DCFME for type float.

## TABLE 5.1.24. - TOW MISSILE POLYNOMIAL DEGREE DATA ARRAY

		1	THEFT	NATA	CSU WHERE SET OR	CSU WHERE USED	DATA SOURCE
NAME OF DATA ELEMENT	DESCRIPTION	MEASURE	VALUE	377	CALCULATED	•	
tow miss noty deaf 0]	polynomial degree for tow missile burn speed		2	int	default declaration miss_tow.c	[miss_tow.c]missile_tow_Init	simnet/data/ms_lw_bs.d
	coefficient data array		,	1	default declaration miss tow.C	[miss tow.c]missile_tow_init	simnet/data/ms_tw_cs.d
tow_miss_poly_deg[ 1]	polynomial degree for tow missile coast speed		<b>1</b>	•	miss tow.chilssile tow init		
	coefficient data array				default declaration miss tow.C.	miss tow.c]missile tow init	simnet/data/me_tw_bt.d
tow_miss_poly_deg[ 2]	polynomial degree for each tow missile burn		•	<b>1</b>	[miss tow.chmissile tow intt	1	
	missile burn turn coefficient data array						
	structure					finites tow clinicalle tow init	simnet/data/ms tw ct.d
tow_miss_poly_deg[ 3]	polynomial degree for each tow missile coast turn coefficient data sub-array of the tow missile coast turn coefficient data array		m	<u> </u>	detault declaration must_cow.c. [muss_tow.c]missile_tow_init		
	structure				The total and an inches to the total	imica tow.clmissile tow init	simnet/data/ms_tw_ct.d
tow_miss_poly_deg[ 4]	NOT USED		•	Ē	inies tow.cimissile_tow_init		

NOTE 1 In It & "C" type for integer.

## TABLE 5.1.25. - TOW MISSILE BURN SPEED COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT	DATA TYPE	CALQUATED	CSU WHERE USED	DATA SOURCE
tow_burn_speed_coeff[ 0]	tow missile burn speed coellident ag; default value 1s 67.0 m/sec	ın/tick	4.46666667	REAL	delault dedaration miss_low.c [miss_tow.c]missile_tow_Init	Indss_tow.c missile_tow_init;   Indss_tow.c missile_tow_fire;   Initss_tow.c missile_tow_fily	simnet/data/ms_tw_bs.d
taw_burn_speed_coeff[ 1]	tow missile burn speed coefficient a;; default value is 274.9732662 m/sec**2	m/lick•2	1.222103405	REAL	default dedaration mise_tow.c [miss_tow.chnissile_tow_int	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fire; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_ts.d
tow_burn_speed_coeff[ 2]	tow missile burn speed coellident az; default value is -82.7057910 m/sec-*3	m/tick**3	-0.024532086	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fire; [miss_tow.c]missile_tow_fiy	simnet/data/ms_tw_bs.d
tow_burn_speed_coeff[ 3]	tow missile burn speed coeflicient ag	m/lick••4	0.0	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fire; [miss_tow.c]missile_tow_fly	simnet/data/ins_tw_bs.d
tow_burn_speed_coeff[ 4]	tow missile burn speed coefficient 24	m/lick**\$	0:0	KEAL	detault dedaration miss tow.c	(nuise_tow.cjmissile_tow_init;  mise_tow.cjmissile_tow_fire;  miss_tow.cjmissile_tow_fly	simnel/data/ms_tw_bs.d

NOTE 1 one titt in equal to one frame or 1/15th of a second NOTE 2 REAL is a "C" mego DEFRE for type float.

## TABLE 5.1.26. - TOW MISSILE COAST SPEED COEFFICIENT DATA ARRAY

NAME OF DATA FLEMENT	DESCRIPTION	UNITS of	DEFAULT	DATA	CSU WHERE SET OR	CSU WHERE USED	DATA SOURCE
		MEASURE INOTE 1	VALUE	TYPE (MOTE 2)	CALCULATED		
tow_coast_speed_coeff[ 0]	tow missile coast speed coefficient ag;	m/tick	21.81905383	REAL	default declaration rulss_tow.c [miss_tow.c]missile_tow_init	(miss_tow.c)missile_tow_init; [miss_tow.chnissile_tow_fly	simnet/data/ms_tw_cs.d
tow_coast_spaed_coeff[ 1]	tow missile coast speed coefficient at;	m/tlck*2	m/tlck*2 -9.5382019e-2	REAL	default declaration miss tow.c	(miss_tow.c)missile_tow_init; [miss_tow.c)missile_tow_fly	simnet/data/ms_tw_cs.d
tow_coast_speed_coeff[ 2]	default value is -21.4609544 m/sec* 7 tow missile coast speed coefficient a 2:	m/tick**3	2.4378222e-4	REAL	default declaration rules tow.c	[miss_tow.c]missile_tow_init; fmiss_tow.chmissile_tow_fly	simnet/data/ms_tw_cs.d
tow_coast_speed_coeff[ 3]	default value is 0.8227650 m/sec**3 tow missile coast speed coefficient a3;	m/lick**4	m/iick**4 -2.6311111e-7	REAL	default dedaration miss low c	[miss_tow.c]missile_tow_init, [miss_tow.c]missile_tow_ily	simnet/data/ms_tw_cs.d
tow_coast_spaed_coeff[ 4]	default value is -0.0133200 m/secc*4 tow missile coast speed coefficient a4	m/tick**5	0:0	REAL	default declaration miss tow.c [miss tow.c]missile tow init	[miss_tow.c]missile_tow_init; [miss_tow.chnissile_tow_fly	simnet/data/ms_tw_cs.d

NOTE 1 one lick in equal to one frame or 1/15th of a second NOTE 2 REAL is a "C" mado DEFRE for type float.

## TABLE 5.1.27. - TOW MISSILE BURN TURN COEFFICIENT DATA STRUCTURE

					SOLITOR COL	CSCI WAFEE LISED	DAIA SOURCE
NAME OF DATA ELEMENT	DESCRPTION	UNITS of MEASURE	DEFAULT	TYPE TYPE	CALCULATED		
		(MO16 1)		Valley WOILE	Carried and an electric to the first of	Imite towchnissile tow init;	simmet/data/ms_tw_bt.d
tow_burn_turn_coeff.deg	polynomial degree for each tow infestie burn turn coefficient data sub-		-		[miss_tow.chulssile_tow_init	[miss_tow.c]missile_tow_fly	
	array of the tow missile burn turn						2
0 33	low missile cosine of maximum side	cos(rad)/lick	0.999976868652	REAL	default declaration miss tow.c	[miss_tow.c]missile_tow_mit;	emmet/data/mb_tw_b.u
tow_burn_turn_coeff.side_coeff vj	turn during burn coefficient ao				miss tow.cgmssue_tow_nut	fulse conceptional low init	nimnet/data/ms tw bl.d
tow burn turn coeff.side_coeff[ 1]	tow missile cosine of maximum side	cos(rad)/Eck**2	-3.5933955e-7	KEAL	(miss tow.chnissile_tow.init	(miss_tow.c]missile_tow_fly	
	turn during burn coefficient at	1 1 1 1 1	0.0000000000000000000000000000000000000	DEAT	default declaration miss tow.c	miss tow.chnissile_tow_init;	simmet/data/ms_tw_bt.d
tow_bum_tum_coeff.up_coeff[ 0]	tow missile cosine of maximum up turn	cos(rad)/ HCK	0.555500001420		Imise tow.chriselle tow init	(miss_tow.c)missile_tow_fly	2 to 1 and
(1) co configuration and more	tow missile cosine of maximum up turn costrad)/tick*2	cox(rad)/lick.2	-3.1492323e-6	REAL	default dedaration miss tow.c	[mise_tow.chniskle_tow_init; [mise_tow.c]miskle_tow_fly	Kitimer/ data/ ille_ra_
	during burn coefficient at	1, 1, 1, 1	0.0000070000000	REAL	default declaration miss tow.c	Inde tow.chnisale tow init	simmet/data/ms_tw_bt.d
tow_burn_turn_coeff.down_coeff[ 0]	tow missile cosine of maximum down	coe(rad)/ ncx	0.33331430101		[mire_tow.chniseile_tow_init	[miss_tow.c]missle_tow_fly	
	turn during burn coefficient at	17,71	7 81040010.0	KI:A1	default declaration miss tow.c	[mise_tow.clinissile_tow_init;	Minnet/dala/mis_lw_ln.d
tow_burn_turn_coeff.down_coeff[ 1]	tow missile cosine of maximum down	T was / must kee			[mlse_tow.c]missile_tow_init	[mlss_tow.c]mlssile_tow_fly	
	turn during burn covilicient #1						

NOTE 1 one let he spial to one frame or 1/15th of a NOTE 2 If At he a "C" mano UNTRE for type final, in a "C" type for integer.

- 399 -

## TABLE 5.1.28. - TOW MISSILE COAST TURN COEFFICIENT DATA STRUCTURE

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE [MOTE 1]	DEFAULT	DATA TYPE (MOIE 2)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
tow_coast_tum_coeff.deg	polynomial degree for each tow missile coast turn coefficient data subarray of the tow missile coast turn coefficient data array structure		3	lnt	detault declaration miss_low.c [miss_tow.c]missdle_tow_init	{miss_tow.c missile_tow_init;  miss_tow.c missile_tow_fly	slunet/data/ms_tw_ct.d
tow_cosst_turn_coeff.side_coeff[ 0]	tow missile cosine of maximum side turn during coast coefficient ap	cos(rad)/tick	0.99995112518	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[mlss_tow.c]missile_tow_init; [mlss_tow.c]missile_tow_fly	simnet/data/ms_tw_ct.d
tow_coast_turn_coeff.side_coeff[ 1]	tow missile cosine of maximum side turn during coast coefficient ay	cos(rad)/tick*2	8.96333e-7	REAL	default dedaration miss_tow.c [miss_tow.c]missle_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnel/data/ins_tw_ct.d
tow_cosst_turn_coeff.side_coeff[ 2]	tow nulssile cosine of maximum side turn during coast coefficient az	cos(rad)/iick*3	-5.995375e-9	REAL	default declaration miss_tow.c [miss_tow.c]misstle_tow_init	[miss_tow.cjmissile_tow_init; [miss_tow.cjmissile_tow_ily	sinnet/data/ms_tw_ct.d
tow_coast_turn_coeff.side_coeff[ 3]	tow missile cosine of maximum side turn during coast coefficient a3	cos(rad)/tick**4	1.162225e-11	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	(miss_tow.c)missile_tow_init; (miss_tow.c)missile_tow_fly	simnet/data/ms_tw_ct.d
tow_coast_tum_coeff.up_coeff[ 0]	tow missile cosine of maximum up turn during coast coefficient an	cos(rad)/tick	0.9998498495	REAL	default dedaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_ct.d
tow_coast_tum_coeff.up_coeff[ 1]	tow missile cosine of maximum up turn during coast coefficient as	cos(rad)/tick**2	1.657779-6	REAL	default declaration miss_tow.c fmiss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_ily	simnet/data/ms_tw_ct.d
tow_coast_tum_coaff.up_coaff[ 2]	tow missile cosine of maximum up turn during coast coeffident a2	cos(rad)/tick**3	-8.231861e-9	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]misslle_tow_init; [miss_tow.c]misslle_tow_fly	simnet/data/ms_tw_ct.d
tow_coast_tum_coaff.up_coaff[ 3]	tow nussile cosine of maximum up turn during coast coefficient a3	cos(rad)/tick**4	1.381832e-11	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_fnit	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_ct.d
taw_coast_tum_coeff.down_coeff[ 0]	tow missile cosine of maximum down turn during coast coeffident ag	cos(rad)/tick	0.9999714014	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_inlt; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_ct.d
tow_coast_tum_coeff.down_coeff[ 1]	tow missile cosine of maximum down turn during coast coefficient at	cos(rad)/lick*2	3.382077e-7	REAL	default declaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_ct.d
tow_coast_tum_coaff.down_coaff[ 2]	tow missile cosine of maximum down turn during coast coefficient a2	cograd)/tick**3	-1.601259e-9	REAL	default dedaration miss_tow.c [miss_tow.c]missile_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_ct.d
tow_cosst_turn_coeff.down_coeff[ 3]	tow missile cosine of maximum down turn during coast coefficient a3	cos(rad)/tick**4	2623014e-12	REAL	default dedaration miss_tow.c [miss_tow.c]missle_tow_init	[miss_tow.c]missile_tow_init; [miss_tow.c]missile_tow_fly	simnet/data/ms_tw_ct.d

MOTE 1

#### TABLE 5.1.29 - ADAT MISSILE CHARACTERISTICS DATA ARRAY

adal_miss_char[ 0] ADAT_BURNOUT_IIME; time of powered  adal_miss_char[ 1] ADAT_BURNOUT_IIME; time of powered  ADAT_BURNOUT_IIME; maximum ADAT_MIX_ATELLIT_IIME; maximum flight time for the adat missile in ticks 130  adal_miss_char[ 2] ADAT_BURNOUT_IIME; maximum flight time for the adat missile in ticks 1300  adal_miss_char[ 3] INVEST_DIST_SQ; default value is 7  meters squared AINE_FUZE_DIST_SQ; default value is 14  meters squared AINE_FUZE_DIST_SQ; default value is 14  meters squared AINE_FUZE_DIST_SQ; default value is 14  AINE_FUZE_DIST_SQ; default value is 14  AINE_FUZE_DIST_SQ; default value is 7  ADAT_TEMP_BIAS_TINE; time of temporal	MEASURE 19 Inote 19 Idea	VALUE	1	CALCULATED		
	þ		NOIE C	,		
	[spuc	48.0	REAL	default declaration miss_adat.c	Imiss_adat.cjmissite_adat_init; Imiss_adat.cjmissite_adat_fly	simnei/data/ms_ad_ch.d
	num ticks ; [20.0	300.00	REAL	default declaration miss_adat.c; [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	sinnet/data/ms_ad_ch.d
HELO, FUZE, DIST, SQ;, delan netes squared AIR_FUZE_DIST_SQ; delault meters squared ADAT_TEMP_BIAS_TINE; in ADAT_TEMP_BIAS_TINE; in	00 meters m*2	0.00006	REAL	default declaration miss_adat.c; [miss_adat.c]missile_adat_init	(miss_adat.c)missile_adat_init; Imiss_adat.c)missile_adat_fly	simnet/data/ms_ad_ch.d
AIR_FUZE_DIST_SQ; default meters squared ADAT_TEMP_BIAS_TIME; un	ie is 7 m°2	49.0	REAL	default declaration miss_adat.c; [miss_adat.c]missile_adat_init	miss_adat.c missile_adat_init;  miss_adat.c missile_adat_fire	sinnet/data/ms_ad_ch.d
ADAT_TEMP_BIAS_TIME; un	Is 14 m°2	196.0	REAL	delault declaration miss_adat.c	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fire	simnet/data/ms_ad_ch.d
Dias for adat missue in ticks [4.0 seconds]	emporal ticks	0.09	REAL	default declaration miss_adat.c	[miss_adat.c]missile_adat_init; [miss_adat.c missile_adat_fly	simnet/data/ms_ad_ch.d
adat_miss_char[ 6] CLOSE_RANCE;	E	2200.0	REAL	default declaration miss_adat.c. [miss_adat.c]missle_adat_init	(miss_adat.c)missile_adat_init; [miss_adat.c)missile_adat_lire	simnet/data/ms_ad_ch.d
adat_miss_char[ 7] NOT USED		0.0	REAL	default declaration miss_adat.c; (miss_adat.c)missile_adat_init	[miss_adat.c]missile_adat_init	simnet/data/ms_ad_ch.d
adat_miss_char( 8) NOT USED		0.0	REAL	default declaration miss adat.c	[miss_adat.c]missile_adat_init	simnet/data/ms_ad_ch.d
adat_miss_char[ 9] NOT USED		0.0	REAL	default declaration miss_adat.c; [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init	simnet/data/ms_ad_ch.d

NOTE 1 one ILI H equal to one frame or 1/15th of a second NOTE 2 REAL H a "C" made DEPME for type float.

#### TABLE 5.1.30. - ADAT MISSILE POLYNOMIAL DEGREE DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT	DATA	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
adat_miss_poly_deg[ 0]	polynomial degree for adat missile burn speed		2	in	default declaration miss_adat.c; [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init	simnet/data/ms_ad_bs.d
adat_miss_poly_deg[ 1]	polynomial degree for adat missile coast speed		4	ţţ	default declaration miss adat.c; [miss adat.c]missile adat int	[miss_adat.c]missile_adat_init	simnet/data/ms_ad_cs.d
edat_miss_poly_deg[ 2]	polynomial degree for cosine of adat missile maximum turn during burn coefficient data		3	in	default dedaration miss_adat.c; [miss_adat.c]missile_adat_int	[miss_adal.c]missile_adal_mit	simnet/data/ms_ad_bt.d
adat_miss_poly_deg[ 3]	paray polynomial degree for cosine of adat missile maximum turn during coast coefficient data		S	ini	default declaration miss_adat.c; [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_hit	simnet/data/ms_ad_ct.d
adat_miss_poly_deg[ 4]	polynomial degree for adat missile temporal bias coefficient data array		4	ĮĮ.	default declaration miss_adat.c; [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init	simnet/data/ms_ad_tb.d

NOTE 1 IN IN 1 C" type for Integer.

## TABLE 5.1.31. - ADAT MISSILE BURN SPEED COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT	DATA TYPE [NOTE 2]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
adat_bum_speed_coeff[ 0 ]	adat niksile burn speed coefficient ag	m/tick	2.2%	REAL	delault declaration miss_adat.c [miss_adat.c]missile_adat_init	Imiss_adal.chnissile_adal_init; Imiss_adal.chnissile_adal_fire; Imiss_adal.chnissile_adal_fly	sinnet/data/ms_ad_bs.d
adat_burn_speed_coeff[ 1]	adat nussile burn speed coellicient a	m/tick*2	0.72990856	KEAL	default dedaration miss_adat.c [miss_adat.c]missle_adat_init	[miss_adal.c]missile_adat_init; [miss_adat.c]missile_adat_fire; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_bs.d
adat_burn_speed_coeff[ 2]	adat missile burn speed coellident a2	m/tick*3	0.013310932	REAL	delault dedaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fire; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_bs.d
sdat_burn_speed_coeff[ 3]	adat mussile burn speed coefficient ag	m/tick**4	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fire; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_bs.d
adst_burn_speed_coeff[ 4]	adat missile burn speed coefficient aq	m/iick**5	0:0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fire; [miss_adat.c]missile_adat_fiy	simnel/data/ms_ad_bs.d
adat_burn_speed_coeff[ 5]	adat miseile burn speed coellident as	m/tick**6	0.0	REAL	default dedaration miss_adat.c [iniss_adat.c]missile_adat_init	[miss_adat.c missile_adat_init; [miss_adat.c missile_adat_fire; [miss_adat.c missile_adat_fiy	elmnet/data/ms_ad_bs.d
adat_burn_speed_coeff[ 6]	adat missile burn speed coefficient ag	m/iick**7	0.0	REAL	default declaration miss_adat.c [miss_adat_c missile_adat_init	imiss_adat.cjmissile_adat_init; [miss_adat.cjmissile_adat_fire; [miss_adat.cjmissile_adat_fiy	simnel/data/ms_ad_bs.d
adat_burn_speed_coeff[ 7]	adat missile burn speed coellident a7	m/tick**8	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_int	miss_adat.c/missile_adat_frut; [miss_adat.c/missile_adat_fre; [miss_adat.c/missile_adat_fr	elmirel/data/ms_ad_bs.q
adat_bum_speed_coeff[ 8]	adal missile burn speed coefficient ag	m/tick**9	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	miss_adat.c}missile_adat_init; [miss_adat.c]missile_adat_fire; [miss_adat.c]missile_adat_fiy	stmnet/data/ms_ad_bs.d
sdat_burn_speed_coeff[ 9]	adat nussile burn speed coellident ag	m/tick**10	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[nuiss_adat.cjrnissile_adat_init; [miss_adat.cjrnissile_adat_fire; [miss_adat.cjrnissile_adat_fiy	simnet/data/ms_ad_bs.d

NOTE 1 one Ikh H equal to one frame or 1/15th of a se NOTE 2 REAL h a "C" mad o DENK for type fout.

## TABLE 5.1.32. - ADAT MISSILE COAST SPEED COEFFICIENT DATA ARRAY

NAME OF DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT	DATA	CALCULATED	LSU WHERE USED	
		NOIL 1		MOIL 7		the territory of the desired or the desired	einnet/data/ms ad cs.d
adat coast speed_coeff[ 0]	adat missile coast speed coefficient ao	m/tick	105.52162	KEAL	default declaration miss_adat.c	inter adat. Chilestic adat. Ily	
					Titles attured to the same	tenter a det charterille adat balt.	cinnet/data/ms ad cs.d
10.00	a last unitarilla conces comme constitucione as	111/11ck**2	-1.0157285	KEAL	delauft declaration miss_adat.c	inisk adole plussine acut, may	
adat_coast_speed_coeff[ 1]	שמשו חווששות בסיטי פלאניו כס חיירים				(misa adat.cliniasile adat hilt	miss adal. Chilastic adal, 119	
		rathly an	5 6124330e-3	KEAL	default declaration miss_adat.c	Indes_adat.chulssile_adat_init;	slinnel/dala/ms_au_cs.u
adat_coast_speed_coeff[ 2]	adal intestie coast speed coefficient at				Imire adat.chuissile adat Init	inisa adat.clmissile adat fly	
		A1.4.04	-1 4242/080-5	REAL	default declaration miss adat c	Imiss adal.chulssile adal init;	simnet/data/ins_ad_cs.d
adat_coast_speed_coeff[ 3]	adat missile coast speed coellicient at	5		_	linies adal cliniestly adat full	inites adat.clinisatic adat fly	
			Contract of		Appendix doctors and and at a	Imiss adat, clinissile adat luit;	simmet/data/ms_ad_cs.d
Ada court cond conff 4]	adat missile coast speed coefficient at	m/lick_5	1.89919326-0	VEAL	The second construction of the second constructi	Instead of stelling and the	
- 1::::::::::::::::::::::::::::::::::::					mis adar. Cimpone adar min	11-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1-1	dimnost/data/me ad Oad
	and the first of the second of	m/tick**6	0.0	REAL	default declaration miss_adat.c	mise adat.cimiseue adat mut,	The state of the s
adat_coast_speed_coeff[ 5]	adat nussiie coast speed corinticut as				imiss adat climissile adat init	(inles adat.clinissile adat Ily	
		100.1.10	2	REAL	default declaration miss adat.c	miss_adat.c)missile_adat_init;	simnel/data/ms_ad_cs.d
adat_coast_speed_coeff[ 6]	adat missile coast speed coefficient a6	, 43m/m	3	?	Imise adat climissile adat init	[miss adat.c]missile_adat_fly	
					Jefer halochealter and at a	Imise adat.chulssile adat init;	simmet/data/ms_ad_cs.d
adat coast speed coeff[ 7]	actat missile coast spayed coefficient ay	m/lick*8	0.0	KEAL	finite adat chaicelle adat init	finise adat.clinisate adat fly	
					Titles minure blackers	lacter attachmissile adat init.	simnet/data/ms ad cs.d
10 337	- 1 - 1 - its court among conflicton so	m/lick*9	0.0	KEAL	default declaration miss adal.c	(introduction of the control of the	
adat_coast_speed_coeff[ 8]	adat missile coast since tocalication as				[miss_adat.c]missile_adat_init	inise adal cimissie adal IIV	
		01. Vil. L 0.10	00	REAL.	default declaration miss adat.c	[miss_adat.c]mlsslle_adat_mit;	simmet/data/ing_ad_co.d
adat_cosst_speed_coeff[ 9]	adat misaile coast speed ewilicient of	2	:		finiss adat.chinissile adat init	Imiss adat.chuissile adat ily	

NOTE 1 one tick is equal to one frame or 1/15th of a 1 NOTE 2 REAL is a "C" macro DEFAE for type float.

## TABLE 5.1.33. - ADAT MISSILE BURN TURN COEFFICIENT DATA ARRAY

ada misele cosine of maximum turn during burn coefficient ag ada misele cosine of maximum turn during burn coefficient ag ada misele cosine of maximum turn during burn coefficient ag ada misele cosine of maximum turn during burn coefficient ag ada misele cosine of maximum turn during burn coefficient ag ada misele cosine of maximum turn during burn coefficient ag ada misele cosine of maximum turn during burn coefficient ag ada misele cosine of maximum turn during burn coefficient ag ada misele cosine of maximum turn during burn coefficient ag ada misele cosine of maximum turn during burn coefficient ag	MEASURE [NOIE 1] COM(rad)/lick COM(rad)/lick**2	VALLE 0.99993	TYPE NOTE 2)	CALCULATED		
adat miselle cosine of maximam turn during burn coefficient an adat miselle cosine of maximam turn during burn coefficient a; adat miselle cosine of maximam turn during burn coefficient a; adat miselle cosine of maximam turn during burn coefficient a; adat miselle cosine of maximam turn during burn coefficient a; adat miselle cosine of maximam turn during burn coefficient a; adat miselle cosine of maximam turn during burn coefficient a; adat miselle cosine of maximam turn during burn coefficient a; adat miselle cosine of maximam turn during burn coefficient as adat miselle cosine of maximam turn during burn coefficient as	cos(rad)/ück cos(rad)/ück*2	0.99993				
adal miselle coshe of maximum turn during burn coefficient a; ada miselle coshe of maximum turn during burn coefficient a; ada miselle coshe of maximum turn during burn coefficient a; adat miselle coshe of maximum turn during burn coefficient a; adat miselle coshe of maximum turn during burn coefficient a; adat miselle coshe of maximum turn during burn coefficient a; adat miselle coshe of maximum turn during burn coefficient a; adat miselle coshe of maximum turn during burn coefficient as	cos(rad)/tick*2		KEAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_liult; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_bt.d
adai missile cosine of maximum turn during burn coefficient az adai missile cosine of maximum turn during burn coefficient az adai missile cosine of maximum turn during burn coefficient az adai missile cosine of maximum turn during burn coefficient az adai missile cosine of maximum turn during burn coefficient az adai missile cosine of maximum turn during burn coefficient az adai missile cosine of maximum turn during burn coefficient az adai missile cosine of maximum turn during burn coefficient ag	_	-6.2386917e-7	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c missile_adat_init; [miss_adat.c missile_adat_fly	simnet/data/ms_ad_bt.d
adat misale cosine of maximum tum during burn coefficient as adat misale cosine of maximum tum during burn coefficient ag adat misale cosine of maximum tum during burn coefficient ag adat misale cosine of maximum tum during burn coefficient as adat misale cosine of maximum tum during burn coefficient as	cos(rad)/fick**3	1.6146426e-7	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_int	[miss_adat.c]missile_adat_Init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_bl.d
adai miselle cosine of maximum tum during burn coefficient ag adat miselle cosine of maximum tum during burn coefficient ag adat miselle cosine of maximum tum during burn coefficient ag during burn coefficient ag	cos(rad)/Eck**4	-9.720142e-7	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	strunel/data/ms_ad_Bc.d
adat mistile cosine of maximum tum during burn coefficient as adat missile cosine of maximum tum during burn coefficient as during burn coefficient as	cos(rad)/Eck**5	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adatc missile_adat_init; [miss_adatc missile_adat_fly	almnet/dals/ms_ad_bk.d
adat mussile cosine of maximum turn during burn coefficient as	cos(rad)/Eck**6	0.0	REAL	default declaration miss_adal.c [miss_adal.c]missile_adal_init	[miss_adat.cjmissile_adat_init; [miss_adat.cjmissile_adat_fly	mmnel/dala/ms_ad_Dt.d
the facility of manipus of the facility of	cos(rad)/lick"7	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_bi.d
nt a7	cos(rad)/tick**8	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	sumnet/data/ms_ad_M.d
imum turn	cos(rad)/fick**9	0.0	REAL	default declaration miss_adatc [miss_adat.c]missile_adat_int	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_ily	simnet/dala/ms_ad_Bcd
mum tum	cos(rad)/lick*10	0.0	REAL	default declaration miss adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_M.d

NOTE 1 one tick in equal to one frame or 1/15th of a vecom NOTE 2 REAL is a "C" made DEFME for type flow.

## TABLE 5.1.34. - ADAT MISSILE COAST TURN COEFFICIENT DATA ARRAY

		MEASURE (NOTE 1)		TYPE (NOTE 2)	CALCULATED		
adat_coast_tum_coeff[ 0]   adai	adat missile cosine of maximum tum durine coast coefficient an	cos(rad)/ück	0.99753111	REAL	default declaration miss_adatc [miss_adatc]missle_adat_init	[miss_adat.c]missile_adat_inlt; [miss_adat.c]missile_adat_fly	slmnet/data/ms_ad_ct.d
#det_cosst_tum_coeff[ 1] adai	adat missile cosine of maximum tum during coast coefficient as	cos(rad)/iick*2	5.5817986e-5	REAL	default declaration miss_adatc [miss_adatc] miss_adatc]missile_adat_mil	[miss_adat.c]missile_adat_Init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_ct.d
adet_coast_tum_coeff[ 2] ada	adat missile cosine of maximum turn during coast coefficient az	cos(rad)/fick**3 -5.1276276e-7	-5.1276276 <del>-</del> 7	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c missile_adat_init; [miss_adat.c missile_adat_fly	simnet/data/ms_ad_ct.d
adat_coast_turn_coeff[ 3] ada	adat missile cosine of maximum tum during coast coeffident a3	cos(rad)/lick**4 2.2338593e-9	2.2338593e-9	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	imiss_adat.chnissile_adat_init; imiss_adat.chnissile_adat_fly	simnet/data/ms_ad_ct.d
adat_coast_tum_coeff[ 4] ada	adat missile cosine of maximum tum during coast coefficient aq	cos(rad)/tick**5 -5.1964622e-12	-5.1964622e-12	REAL	default declaration miss_adat.c [miss_adat.c]misslle_adat_init	(miss_adat.c)missile_adat_init; (miss_adat.c)missile_adat_fly	simnet/data/ms_ad_ct.d
adat_coast_tum_coeff[ 5] ada	adat missile cosine of maximum tum during coast coefficient as	cos(rad)/Eck**6 4.5/99104e-15	4.5499104e-15	REAL	default declaration miss_adatc [miss_adatc]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_ct.d
adat_cosst_tum_coeff[ 6] ada	adat missile cosine of maximum tum during coast coefficient a6	cos(rad)/lick**7	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnel/data/ms_ad_ct.d
adat_coast_tum_coaff[ 7] ada	adat missile costne of maximum tum during coast coefficient a7	cos(rad)/tick**8	0.0	REAL	default declaration miss_adat.c [miss_adat.c]missile_adat_init	miss_adat.c missile_adat_int;  miss_adat.c missile_adat_fly	sunnet/data/ms_ad_ct.d
adet_cosst_tum_coeff[ 8] ada	adat nussile cosine of maximum tum during coast coefficient ag	cos(rad)/lick**9	0.0	REAL	default declaration miss_adat.c [miss_adat.c missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_ily	simnet/data/ms_ad_ct.d
adat_coast_tum_coeff[ 9] ada	adat missile cosine of maximum tum during coast coefficient ao	cos(rad)/lick**10	0.0	REAL	default declaration miss_adat.c {miss_adat.c missile_adat_init	[miss_adat.c]missile_adat_init; [miss_adat.c]missile_adat_fly	simnet/data/ms_ad_ct.d

NOTE 1 ove lick is equal to one frame or 1/15th of a seco-NOTE 2 REAL is a "C" mad o DEFINE for type floid.

**-** 405 -

## TABLE 5.1.35. - ADAT MISSILE TEMPORAL BIAS COEFFICIENT DATA ARRAY

NAME OF DATA ELEMENT	DESCRIPTION	UNITS of	DEFAULT	DATA	CSU WHERE SET OR	CSU WHERE USED	DATA SOURCE
		(HOIE 1)	* ALUE	[NOTE 2]	Carcalan		
adet temo bise coaff (1)	adat missile temporal bias coefficient an	cos(rad)/lick	5.3105657e-2	KEAL	default declaration miss_adat.c	(miss_adat.c missile_adat_init;	simnet/data/ms_ad_ct.d
					[miss_adat.c]missile_adat_init	(miss adat.c)missife adat fly	
the confit 11	adat missile temporal hiss coefficient at	costrad)/lick**2 7.1795817e-2	7.1795817e-2	REAL	default declaration miss adat.c	[miss_adat.chnissile_adat_init;	simnet/data/ms_ad_ct.d
i. lippalator dupa lang					[miss_adat.c]missile_adat_mit	imiss adat.chnissile_adat_fly	
adat temp bise coaff 21	adat missile temporal bias coefficient as	costrad)/lick*3	1.80346466-2	KEAL	default declaration miss_adat.c	Inuss_adat.c missile_adat_init;	simnel/data/ms_ad_ct.d
	4				Indss adat.cimissife adat init	Imiss adat.c)missile adat fly	
ades terms bies confif 31	adat missile temporal hias coefficient an	cos(rad)/lick**4 -6.0083762e-4	-6.0083762e-4	REAL	default declaration miss_adat.c	[mics_adat.c]missile_adat_init;	sinnet/data/ins_ad_ct.d
					imise adat.c miselle_adat_init	Intse adat clinisatic adat fly	
ades tome him conff 41	adat missile termonal hias coefficient as	costrad)/lick**5	4.67610916-6	REAL	default declaration miss_adat.c	(miss_adat.c)missile_adat_init;	sinnet/data/ms ad ct.d
					[miss_adat_c]nussile_adat_init	[miss_adat.c]missile_adat_fly	
the Arms blue acade 51	adal missile temawal bias coefficient as	costrad)/lick**6	0.0	REAL	default declaration miss_adal.c	[miss_adat.c)missile_adat_init;	simnet/data/ms_ad_ct.d
fc linear-countries					Indes adat climissile adat init	Imiss adat.c]missle_adat_fly	
12. 17. 17. 17. 17. 17. 17.	adat missile termonal bias coefficient ac	costrad)/tick*7	0.0	REAL	default declaration miss_adat.c	[miss_adat.c]missile_adat_init;	simnet/data/ms_ad_ct.d
soar remb_case_coart of		:			(miss adat.c]missile_adat_init	Iniss adat.c)missile adat fly	
. de	adat missile temascal bias coefficient at	costrad)/lick*8	0.0	KEAL	default declaration miss_adat.c	Indes adat.chnissile_adat_init;	shnnet/data/ms_ad_ct.d
במון ופנולה ואפריכים ול ין					[miss_adat.c]missile_adat_init	(miss_adat.c missile_adat_fly	
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	at at unfecile temporal hias coefficient as	cos(rad)/tick**9	0.0	REAL	default declaration miss adat.c	[miss_adat.chnissile_adat_init;	sinnet/data/ms_ad_ct.d
Boat temp man could be					[miss_adat.c]missile_adat_init	Imiss adat.c)missile adat fly	
10 3/100 001	selat infesile temparal bias coefficient ao	cos(rad)/lick*10	0.0	REAL	default declaration mas adat.c	Indes adat clinissile adat init;	stumet/data/ms_ad_cl.d
adat_temp_data_count_3					Imiss adat cludssile adat lult	Imiss_adat.chulssile_adat_fly	

NOTE 1 one tick is equal to one frame or 1/15th of a se NOTE 2 REAL is a "C" macso DERMC for type float.

- 406 -

#### TABLE 5.1.36 - ATGM MISSILE CHARACTERISTICS DATA ARRAY

NAME OF DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT VALUE	DATA TYPE INOIT 2]	CALCALATED	CSU WHERE USED	DATA SOURCE
tow_miss_char[ 0]	10W_BURNOUT_11MI; time of powered flight for tow missile in ticks [1.6 seconds]	licks	24.0	KEAL	default declaration miss_atgm.c; [miss_atgm.c]missile_atgm_init	Integ atgm.clndesile_atgm_int; integration of the content of the c	simmet/data/ms_at_dr.d
tow_miss_char[ 1]	TOW_RANGE_LIMIT_TIME range limit time for the tow missile in ticks [17,80 accounts]; at this point the wire is cut, but the missile in allowed to five to the maximum flight time	ücks	268.35	REAL	default declaration miss_atgm.c; [miss_atgm.c]missile_atgm_init	Imiss_atgm.c]missite_atgm_init; Imiss_atgm.c}missite_atgm_ify	simnet/data/ms_at_ch.d
tow_miss_char[ 2]	TOW_MAX_FLIGHT_TIME; maximum Hight time for the tow missile in theks, excine of the max turn is greater than 1.0 beyond this point	ticks	200:00	REAL	default declaration miss_atgm.c; [miss_atgm.chnissile_atgm_init	(miss_atgm.c]misslie_atgm_init	simnet/data/ms_at_ch.d
tow_miss_char[ 3]	ATCM_TURN_FACTOR; ATCM turn lactor for wider turning capability with respect to TOW		6'0	REAL	default declaration miss_atgm.c; (miss_atgm.c)misstle_atgm_init	[nuss_atgm.c]mussile_atgm_Init	simnet/data/ms_at_ch.d
tow_miss_char[ 4]	NOI USED		0.0	KEAL	delauli dedaration intes atgm.c [miss_atgm.chnissile_atgm_int	Indss_aigm.c]missile_aigm_init	simmet/data/ms_at_dr.d

NOTE 1 one list in equal to me frame or 1715th of a second NOTE 2 REAL IS 8"C" metro DEPRE for type fout.

### TABLE 5.1.37. - ATGM MISSILE POLYNOMIAL DEGREE DATA ARRAY

NAME OF DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT	DATA	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
tow_miss_poly_deg[ 0]	polynomial degree for low missile burn speed		2	int int	default declaration miss_aigm.c; [miss_aigm.c]missle_aigm_init	[miss_atgm.chmissile_atgm_init	sinnet/data/ms_at_bs.d
tow_miss_poly_deg[ 1]	polynomial degree for low missile coast speed		c	Į.	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.chnissile_atgm_init	simmet/data/ms_at_cs.d
tow_miss_poly_deg[ 2]	polynomial degree for each tow missile burn turn coefficient data sub-array of the low missile burn turn coefficient data array		-	<u>14</u>	default declaration miss_atgm.c; [miss_atgm.c]missile_atgm_init	miss_atgm.chotssile_atgm_init	simnet/data/ms_at_bt.d
tow_miss_poly_deg[ 3]	polynomial degree for each tow missile coast turn coefficient data sub-array of the tow missile coast turn coefficient data array		e a	ī	delault declaration miss_stgm.c [miss_stgm.c]missile_atgm_init	iniss_stgn.chissile_stgn_init	simnet/data/ms_at_ct.d
tow_miss_poly_deg[ 4]	NOT USED		0	int	default declaration miss_atgm.c; [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init	simnet/data/ms_at_ct.d

NOTE 1 In Is a "C" type for ini

## TABLE 5.1.38. - ATGM MISSILE BURN SPEED COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of	DEFAULT	DATA	CSU WHERE SET OR	CSU WHERE USED	DATA SOURCE
	•	MEASURE [NOIT 1]	VALLÆ	TYPE [MOTE 2]	CALGULATED		
tow harn sneed coafff 01	low missile burn speed coefficient ao:	ın/tick	4.466666667	REAL	default declaration miss_atgm.c	[mise_atgm.c mlasile_atgm_Indt;	sinnet/data/ms_at_bs.d
	default value is 67.0 m/sec				[miss_atgm.c]missile_atgm_init	Imise_atgm.clmissile_atgm_fire;	
						inuss argm.cimissue argin ity	
tow burn speed coeff 11	tow missile burn spood coefficient at;	m/lick*2	1.222103405	KEAL	default declaration miss_atgm.c	[miss_algm.c]missile_algm_init;	sinnel/dala/ms_st_bs.d
	default value is 274.9732662 m/sec*2				(miss_atgm.c)missile_atgm_init	[miss_atgm.c]misstle_atgm_fire;	
						(miss_atgm.chnissile_atgm_fly	
tow have speed coeff 21	tow missile burn streed coefficient an:	m/tick*3	-0.024532086	REAL	default declaration miss_algm.c	(mise_atgm.c)missile_atgm_init;	simnet/data/ms_at_bs.d
- Image de la company de la co	default value is -82 7057910 m/sec**3				[miss_atgm.c]missile_atgm_init	(miss_atgm.c]missile_atgm_fire;	
						Imiss_aigm.chnissile_aigm_fly	
tou hear speed metil 31	tow missile burn speed coefficient as	m/tick**4	0.0	REAL	default declaration miss_atgm.c	[miss_atgm.c]missile_atgm_init;	simnet/data/ms_at_bs.d
In Times Transfer and Temperature					[miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_fire;	
						Iniss atgm.c)missile atgm fly	
tow hom seem coaff 41	tow missile burn speed coefficient as	m/tick**5	0:0	KEAL	default declaration miss_atgm.c	(miss_aigm.c)missile_aigm_init;	simnet/data/ms_at_bs.d
					[miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_fire;	
					•	Imiss atgm.chnissile atgm fly	

NOTE 1 one lick in equal to one frame or 1/15th of a second NOTE 2 REAL is a "C" mean DEPRE for type float.

## TABLE 5.1.39. - ATGM MISSILE COAST SPEED COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS OF MEASURE	DEFAULT	DATA TYPE	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
tow_coast_speed_coeff[ 0]	tow missile coast speed coefficient ag: default value is 377.2858074 m/sec	m/tick	21.81905383	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]mdssile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_cs.d
tow_coast_spaed_coeff[ 1]	tow missile coast speed coefficient at; default value is -21,4609544 m/sec**2	m/lick*2	m/lick*2 -9.5382019e-2	REAL	default dedaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_cs.d
tow_cosst_speed_coeff[ 2]	tow missile coast speed coefficient ag. default value is 0.8227650 m/sec*3	m/tlck*3	m/lick**3 24378222e-4	KEAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_int	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_os.d
tow_coast_speed_coeff[ 3]	tow missile coast speed coefficient a3:	m/tick**4	m/tick**4 -26311111e-7	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_cs.d
tow_coast_speed_coeff[ 4]	tow missile coast speed coefficient at	m/tick**5	0.0	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	sinunet/data/ms_al_cs.d

NOTE 1 one lick is equal to one frame or 1/15th of a second NOTE 2 REAL is a "C" mado DEFNE for type float.

## TABLE 5.1.40. - ATGM MISSILE BURN TURN COEFFICIENT DATA STRUCTURE

NAME Of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE (MOTE 1)	DEFAULT	DATA TYPE (NOTE 2)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
tow_burn_turn_coeff.deg	polynonial degree for each tow missile burn turn coefficient data subarray of the tow missile burn turn coefficient data array structure		-	ī	default dedaration miss_atgm.c [miss_atgm.c]missile_atgm_init	Indss_atgm.c missile_atgm_init; [iniss_atgm.c missile_atgm_fly	sinuici/data/ms_al_bt.d
tow_burn_turn_coeff.side_coeff[ 0]	tow missile cosine of maximum side	cos(rad)/lick	0.999976868652	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	(miss_atgm.c missile_atgm_inti; [miss_atgm.c missile_atgm_fly	simnet/data/ms_at_bt.d
tow_burn_turn_coeff.side_coeff[ 1]	tow missile cosine of maximum side	cos(rad)/iick**2	-3.59339556-7	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_ily	simnet/data/ms_at_bt.d
tow_burn_turn_coeff.up_coeff[ 0]	tow missile cosine of maximum up turn during burn coefficient an	cos(rad)/tick*	0.999960667258	REAL	default declaration mas_atgm.c [miss_atgm.c]missile_atgm_init	Imiss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	
tow_bum_tum_coeff.up_coeff[ 1]	tow missile cosine of maximum up turn during burn coefficient a	cos(rad)/lick*2	-3.1492328e-6	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_int	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	
tow_burn_turn_coeff.down_coeff[ 0]	tow missile cosine of maximum down turn during burn coefficient as	cos(rad)/tick	0.999978909989	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	miss_atgm.c missile_atgm_init;  miss_atgm.c missile_atgm_fly	simnet/dala/ms_al_bt.d
tow_burn_turn_coeff.down_coeff[ 1]	tow missile cosine of maximum down	cograd)/iick*2	6-91661618'-2-	KEAL	default declaration miss_atgm.c imiss_atgm.cjmissile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simmet/data/ms_at_bk.d

NOTE 1 one first in round to one frame or 1/13th of a second NOTE 2 REAL IN a "C" mano UN HM for type fow.

Int in a "G" type for integer.

# TABLE 5.1.41. - ATGM MISSILE COAST TURN COEFFICIENT DATA STRUCTURE

NAME Of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE [ROIT 1]	DEFAULT	DATA TYPE [MOIT 2]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
tow_cosst_tum_cosff.deg	polynomial degree for each tow missile coast turn coefficient data subarray of the tow missile coast turn coefficient data array structure		3	ht	default dedaration miss_aigm.c [miss_aigm.c missile_aigm_init	{miss_atgm.c misstle_atgm_init;  miss_atgm.c misstle_atgm_fly	simmet/data/ms_at_ct.d
tow_coast_turn_coeff.side_coeff[ 0]	tow missile cosine of maximum side	cos(rad)/lick	0.99995112518	REAL	default declaration miss_atgm.c [niss_atgm.c]missile_atgm_init	(miss_atgm.cjmissile_atgm_init; [miss_atgm.cjmissile_atgm_fly	simnet/data/ms_al_ct.d
tow_coast_tum_coeff.side_coeff[ 1]	tow nukile cosine of maximum side turn during coast coeffident a	costrad)/lick*2	8.96333e-7	KEAL	default declaration miss_atgm.c [miss_atgm.c]misstle_atgm_init	Indes_atgm.c]missile_atgm_init; [miss_atgm.chnissile_atgm_fly	simnet/data/ms_al_ct.d
tow_coast_turn_coeff.side_coeff[ 2]	tow missile cosine of maximum side	cos(rad)/Eck**3	-5.995375e-9	REAL	delault declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	(miss_atgm.c)missile_atgm_init; (miss_atgm.c)missile_atgm_fly	simmet/data/ms_at_ct.d
tow_coast_turn_coeff.side_coeff[ 3]	tow missile cosine of maximum side	cograd)/Eck**4	1.162225e-11	REAL	default declaration miss_atgm.c [miss_atgm.c]missüe_atgm_init	Indsa_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	sinnet/data/ms_at_ct.d
tow_coast_tum_coeff.up_coeff[ 0]	tow raissile cosine of maximum up turn during coast coeffident ag	cos(rad)/tick	0.9998498495	REAL	default dedaration miss_atgm.c [miss_atgm.c]missile_atgm_intt	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simmet/data/ms_st_ct.d
tow_coast_tum_coeff.up_coeff[ 1]	tow missile cosine of maximum up turn during coast coefficient at	cos(rad)/tick**2	1.657779e-6	REAL	default declaration miss_atgm.c [miss_atgm.chnissile_atgm_init	(miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	sinuet/data/ms_at_ct.d
tow_coast_tum_coeff.up_coeff[ 2]	tow missile cosine of maximum up turn during coast coefficient a2	cos(rad)/lick**3	-8.231861e-9	REAL	default dedaration miss_atgm.c [miss_atgm.c]missile_atgm_init	miss_atgm.cjmissile_atgm_init;  iniss_atgm.cjmissile_atgm_fly	simnel/data/ms_at_ct.d
tow_coast_tum_coeff.up_coeff[ 3]	tow intestle cosine of maximum up turn during coast coefficient as	cos(rad)/Eck**4	1.381832e-11	KEAL	delault dedaration miss_atgm.c [miss_atgm.chnisslie_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simmet/data/ins_at_ct.d
tow_coast_tum_coeff.down_coeff[ 0]	tow missile cosine of maximum down turn during coast coefficient ag	cos(rad)/tick	0.9999714014	REAL	default declaration miss_atgm.c [miss_atgm.c]missile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnet/data/ms_at_ct.d
tow_const_turn_coeff.down_coeff[ 1]	tow missile costne of maximum down turn during coast coefficient at	cos(rad)/tick*2	3.382U77e-7	REAL	default declaration miss_atgm.c [miss_atgm.chnissile_atgm_init	[miss_atgm.c]missile_atgm_init; [miss_atgm.c]missile_atgm_fly	simnel/data/ins_at_ct.d
tow_coast_tum_coeff.down_coeff[ 2]	tow missile cosine of maximum down turn during coast coefficient a2	cos(rad)/Eck*3	-1.601259æ9	REAL	default dedaration miss_atgm.c [miss_atgm.c]missile_atgm_init	(miss_atgm.c]missile_atgm_init; (miss_atgm.c)missile_atgm_fly	simnet/data/ms_al_ct.d
tow_coast_tum_coeff.down_coeff[ 3]	tow missile cosine of maximum down turn during coast coefficient as	cos(rad)/tick**4	2623014e-12	REAL	default dedaration miss_atgm.c {miss_atgm.c}missile_atgm_init	(miss_atgm.chnisstle_atgm_init; [miss_atgm.chnisstle_atgm_fly	simmet/data/ms_at_ct.d

NOTE 1 one tick is equal to one frame or 1/15th of a se NOTE 2 REAL is a "C" made DEPME for type float.

- 410 -

#### TABLE 5.1.42 - KEM MISSILE CHARACTERISTICS DATA ARRAY

NAME OF DATA ELEMENT	DESCRIPTION	UNITS of MEASURE [HOIT 1]	DEFAULT	DATA TYPE [HOIL ?]	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
kem_miss_char[ 0]	KEM_BURNOUT_TIME; time of powered flight for kem missile in ticks [3.2 seconds]	licks	48.0	REAL	delault declaration miss_kem.c; fmiss_kem.cjmiestle_kem_init	Iniss_kern.clmissile_kern_lidt; Iniss_kern.clmissile_kern_liy	որուլ/վուո/ms_km_dr.d
kem_miss_char[ 1]	KEM_MAX_FLIGHT_TIME; maximum filight time for the kem missile in ticks [20,0 seconds]	ücks	300:00	REAL	default declaration miss_kem.c; [miss_kem.c]missile_kem_init	Indes kem.c]ndssile_kem_init; Imiss_kem.c]missile_kem_ily	simmet/data/ms_km_ch.d
kern_miss_char[ 2]	KEM_TO_MACHS_FACTOR; speed factor to raise from ADAT to KEM; just after burnout, the ADAT has a maximum velocity of 230 m/sec, while the KEM has a maximum velocity of 1524 m/sec.		6.626	REAL	default dedaration miss_kem.c; [miss_kem.c}missile_kem_init	Imiss_kem.chnissile_kem_init; Imiss_kem.chnissile_kem_ily Imiss_kem.chnissile_kem_ily	stinnet/data/ins_kin_ch.d
kem_miss_char[ 3]	NOT USED		0.0	REAL	default declaration miss_kem.c; [miss_kem.c]missile_kem_init	indss_kem.cjmissile_kem_indt; imiss_kem.cjmissile_kem_iire	simnet/data/ms_km_ch.d
kem_miss_char[ 4]	NOT USED		0.0	KEAL	default declaration mise_kem.c; finise_kem.c)missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fire	simnet/data/ms_km_ch.d
kem_miss_char{ 5}	NOT USED		0.0	REAL	default declaration miss kem.c. Imiss kem.chnissile kem intt	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_ily	simnet/data/ms_km_ch.d
kem_miss_char[ 6]	NOI USED		0.0	KEAL	delault dedaration miss_kem.c; [nulss_kem.c]missile_kem_init	Indes_kem.cjnussile_kem_int; Imiss_kem.cjmissile_kem_fire	simnet/data/ms_km_ch.d
kem_miss_char[ 7]	NOT USED		0.0	REAL	default declaration miss_kem.c; (miss_kem.c)missile_kem_init	Inuss_kem.c]missile_kem_init	sinnet/data/ms_km_ch.d
korn_miss_char[ 8]	NOT USED		0.0	REAL	default declaration miss_kem.c; [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init	simnet/data/ms_km_ch.d
kem_miss_char[ 9]	NOT USED		0.0	REAL	default declaration miss kem.c. [miss kem.chnissile kem Init	[mlss_kem.c]missile_kem_init	simnet/data/ms_km_ch.d

VOTE 1 one tick is repaid to one frame or 1/15th of a second 401E.2 REAL is a "C" macro DEFRE for type float.

#### TABLE 5.1.43. - KEM MISSILE POLYNOMIAL DEGREE DATA ARRAY

	DESCRIPTION	UNITS OF MEASURE	NALLE	DATA TYPE [MOTE 1]	CSU WIERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
kem_miss_poly_deg[ 0]	polynomial degree for kem missile burn speed coefficient data array		7	int	delault declaration miss_kem.c; [miss_kem.c]missile_kem_indt	Imies_kem.c]miesile_keın_init	simnet/data/ms_km_bs.d
kem_miss_poly_deg[ 1]	polynomial degree for kem missile coast speed		7	Į.	default dedaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init	simnet/data/ms_km_cs.d
kem_miss_poly_deg[ 2]	polynomial degree for cosine of kein missile maximum turn during burn coefficient data array		£.	int	default declaration miss_kem.c; [miss_kem.c]nussile_kem_init	[miss_kem.c]missile_kem_init	sinnet/data/ms_km_bt.d
kem_miss_poly_deg[ 3]	polynomial degree for cosine of kem missile maximum turn during coast coeffident data array		15	int	default declaration miss_kem.c; [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init	simnet/data/ms_km_ct.d
kem_miss_poly_deg[ 4]	NOT USED		0	int	default declaration miss kem.c		

1 in its a "C" type for integer.

## TABLE 5.1.44. - KEM MISSILE BURN SPEED COEFFICIENT DATA ARRAY

2 0.72990856 2 0.72990856 3 0.013310932 4 0.0 6 0.0 7 0.0 9 0.0 10 0.0	NAME OF DATA ELEMENT	DESCRIPTION	UNITS of	DEFAULT	DATA	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
kem missile burn speed coellident 20 m/tick** 25% REAL default declaration miss. kem. cliniss. kem.			(NOTE 1)		NOTE 2]		11 21 21 21 21 21 21 21 21 21 21 21 21 2	ofmnet /data /ms km had
kem missile burn speed coefficient 2 m/tick**7 0013310932 REAL default declaration miss. kem. c li miss. kem. c lideault declaration miss. kem	kem_bum_speed_coeff[ 0]		m/tick	2.2%	REAL	detault dedaration miss_kem.c fmiss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_fire;	
kem missile burn speed coellident a 1 m/itck**2 0.72990556 REAL default declaration miss. kem.c   Imits kem.c   Im							[miss kem.c]missile kem fly	F 24 7 7 1
kem missile burn speed coefficient as m/tick**3 0.013310932 REAL default declaration miss_kemc_imissile burn speed coefficient as m/tick**4 0.0 REAL default declaration miss_kemc_imissile burn speed coefficient as m/tick**4 0.0 REAL default declaration miss_kemc_initsile burn speed coefficient as m/tick**6 0.0 REAL default declaration miss_kemc_initsile burn speed coefficient as m/tick**7 0.0 REAL default declaration miss_kemc_initsile burn speed coefficient as m/tick**7 0.0 REAL default declaration miss_kemc_initsile burn speed coefficient as m/tick**7 0.0 REAL default declaration miss_kemc_initsile burn speed coefficient as m/tick**7 0.0 REAL default declaration miss_kemc_initsile burn speed coefficient as m/tick**7 0.0 REAL default declaration miss_kemc_initsile burn speed coefficient as m/tick**7 0.0 REAL default declaration miss_kemc_inits_kemc_initsile burn speed coefficient as m/tick**7 0.0 REAL default declaration miss_kemc_inits_inits_kemc_inits_inits_kemc_inits_kemc_inits_inits_kemc_inits_inits_kemc_inits_inits_kemc_inits_inits_inits_kemc_inits_kemc_inits_inits_kemc_inits_inits_kemc_inits_kemc_inits_inits_kemc_inits_kemc_inits_kemc_inits_kemc_inits_kemc_inits_kemc_inits_kemc_inits_kemc_inits_kemc_inits_kemc_inits_kemc_inits_inits_inits_kemc_inits		100 100	Cataly as	0.72990856	REAL	default declaration miss_kem.c	[miss_kem.c]missile_kem_init;	simnet/data/ms_xm_ts.u
kem missile burn speed coefficient ag m/tick*** 0.0013310932 REAL default declaration miss. kem.c missile burn speed coefficient ag m/tick*** 0.00 REAL default declaration miss. kem.c missile burn speed coefficient ag m/tick*** 0.00 REAL default declaration miss. kem.c missile burn speed coefficient ag m/tick*** 0.00 REAL default declaration miss. kem.c missile burn speed coefficient ag m/tick*** 0.00 REAL default declaration miss. kem.c missile burn speed coefficient ag m/tick*** 0.00 REAL default declaration miss. kem.c missile burn speed coefficient ag m/tick*** 0.00 REAL default declaration miss. kem.c missile burn speed coefficient ag m/tick*** 0.00 REAL default declaration miss. kem.c missile burn speed coefficient ag m/tick*** 0.00 REAL default declaration miss. kem.c missile burn speed coefficient ag m/tick*** 0.00 REAL default declaration miss. kem.c missile burn speed coefficient ag m/tick*** 0.00 REAL default declaration miss. kem.c missile burn speed coefficient ag m/tick*** 0.00 REAL default declaration miss. kem.c missile burn speed coefficient ag m/tick*** 0.00 REAL default declaration miss. kem.c missile burn speed coefficient ag m/tick*** 0.00 REAL default declaration miss. kem.c missile burn speed coefficient ag m/tick*** 0.00 REAL default declaration miss. kem.c missile burn speed coefficient ag m/tick*** 0.00 REAL default declaration miss. kem.c missile burn speed coefficient ag m/tick*** 0.00 REAL default declaration miss. kem.c missile burn speed coefficient ag m/tick**** 0.00 REAL default declaration miss. kem.c missile burn speed coefficient ag m/tick************************************	kem_bum_speed_coeff[ 1]	kem missue burn speca coeuncem at	4 51			[miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_lire;	
kem missile burn speed coefficient as m/tick**3 0013310932 REAL default declaration miss. kem.c missile burn speed coefficient as m/tick**5 0.0 REAL default declaration miss. kem.c missile burn speed coefficient as m/tick**5 0.0 REAL default declaration miss. kem.c missile burn speed coefficient as m/tick**5 0.0 REAL default declaration miss. kem.c missile burn speed coefficient as m/tick**5 0.0 REAL default declaration miss. kem.c missile burn speed coefficient as m/tick**7 0.0 REAL default declaration miss. kem.c missile burn speed coefficient as m/tick**8 0.0 REAL default declaration miss. kem.c missile burn speed coefficient as m/tick**8 0.0 REAL default declaration miss. kem.c missile burn speed coefficient as m/tick**8 0.0 REAL default declaration miss. kem.c missile burn speed coefficient as m/tick**8 0.0 REAL default declaration miss. kem.c missile burn speed coefficient as m/tick**8 0.0 REAL default declaration miss. kem.c missile burn speed coefficient as m/tick**9 0.0 REAL default declaration miss. kem.c missile burn speed coefficient as m/tick**9 0.0 REAL default declaration miss. kem.c missile burn speed coefficient as m/tick**9 0.0 REAL default declaration miss. kem.c missile burn speed coefficient as m/tick**10 0.0 REAL default declaration miss. kem.c missile burn speed coefficient as m/tick**10 0.0 REAL default declaration miss. kem.c missile burn speed coefficient as m/tick**10 0.0 REAL default declaration miss. kem.c missile burn speed coefficient as m/tick**10 0.0 REAL default declaration miss. kem.c missile burn speed coefficient as m/tick**10 0.0 REAL default declaration miss. kem.c missile burn speed coefficient as m/tick**10 0.0 REAL default declaration miss. kem.c missile burn speed coefficient as m/tick**10 0.0 REAL default declaration miss. kem.c missile burn speed coefficient as m/tick**10 0.0 REAL default declaration miss. kem.c missile burn speed coefficient as m/tick**10 0.0 REAL default declaration miss. kem.c missile burn speed coefficient as m/tick**10 0.0 Missile burn speed coe							miss kem.cjimssije kein iiy	1 24 ml 2 ml 1 ml
kem missile burn speed coellident as m/tick**4 0.0 REAL default dedaration miss kem.cjmissile kem_init in missile burn speed coellident as m/tick**5 0.0 REAL default dedaration miss kem.cjmissile kem_init in kem missile burn speed coellident as m/tick**5 0.0 REAL default dedaration miss kem.c inits kem.cjmissile kem_init inits kem missile burn speed coellident as m/tick**5 0.0 REAL default dedaration miss kem.c inits kem.cjmissile kem_init inits kem.cjmissile kem.			F**4511/ m	0.013310932	REAL	default declaration miss_kem.c	miss kem.cjmissile kem_int;	Simnet/data/ins_kin_os.u
kem missile burn speed coellident ag m/tick*4 0.00 REAL default dedaration miss. kem.c missile burn speed coellident ag m/tick*5 0.00 REAL default dedaration miss. kem.c missile burn speed coellident ag m/tick*6 0.00 REAL default dedaration miss. kem.c missile burn speed coellident ag m/tick*7 0.00 REAL default dedaration miss. kem.c missile burn speed coellident ag m/tick*8 0.00 REAL default dedaration miss. kem.c missile burn speed coellident ag m/tick*8 0.00 REAL default dedaration miss. kem.c missile burn speed coellident ag m/tick*9 0.00 REAL default dedaration miss. kem.c missile burn speed coellident ag m/tick*9 0.00 REAL default dedaration miss. kem.c missile burn speed coellident ag m/tick*9 0.00 REAL default dedaration miss. kem.c missile burn speed coellident ag m/tick*10 0.00 REAL default dedaration miss. kem.c missile burn speed coellident ag m/tick*10 0.00 REAL default dedaration miss. kem.c missile burn speed coellident ag m/tick*10 0.00 REAL default dedaration miss. kem.c missile burn speed coellident ag m/tick*10 0.00 REAL default dedaration miss. kem.c missile burn speed coellident ag m/tick*10 0.00 REAL default dedaration miss. kem.c missile burn speed coellident ag m/tick*10 0.00 REAL default dedaration miss. kem.c missile burn speed coellident ag m/tick*10 0.00 REAL default dedaration miss. kem.c missile burn speed coellident ag m/tick*10 0.00 REAL default dedaration miss. kem.c missile burn speed coellident ag m/tick*10 0.00 REAL default dedaration miss. kem.c missile burn speed coellident ag m/tick*10 0.00 REAL default dedaration miss. kem.c missile burn speed coellident ag m/tick*10 0.00 REAL default dedaration miss. kem.c missile burn speed coellident ag m/tick*10 0.00 REAL default dedaration miss. kem.c missile burn speed coellident ag m/tick*10 0.00 REAL default dedaration miss. kem.c missile burn speed coellident ag m/tick*10 0.00 REAL default dedaration miss. kem.c missile burn speed coellident ag m/tick*10 0.00 Missile burn speed coellident ag m/tick*10 0.00 Missile burn speed coel	kem_bum_speed_coeff[ 2]		,			[miss_kem.c]missile_kem_init	(miss kem.c)missile kem lire;	
kem missile burn speed coellident 23 m/lick**4 0.0 REAL default dedaration miss. kem.c missile burn speed coellident 24 m/lick**5 0.0 REAL default dedaration miss. kem.c missile burn speed coellident 25 m/lick**5 0.0 REAL default dedaration miss. kem.c limis. kem.c missile burn speed coellident 25 m/lick**7 0.0 REAL default dedaration miss. kem.c limis.							imiss Kein Chiussie Kein it	elmnet /data /ms km bs.d
kem missile burn speed coeliident as m/tick**5 0.0 REAL default dedaration miss kem.clmissile kem.lnti liniss_kem.clmissile kem.lntiss_kem.clmissile kem.lntiss_kem.clmissile kem.lntiss_kem.clmissile kem.clmissile kem.lntiss_kem.clmissile kem.lntiss_kem.clmissile kem.clmissile kem.clmissile kem.clmissile kem.lntiss_kem.clmissile kem.clmissile kem.clmissile kem.clmissile kem.clmissile kem.clmissile kem.clmissile kem.clmissile kem.lntiss_kem.clmissile kem.clmissile kem.c			m /tick**4	0.0	REAL	default declaration miss_kem.c	mee kem.clinissile kem ilut.	man /
kem missile burn speed coellident 24 m/tick**5 0.0 REAL default dedaration miss kemc links kemc missile burn speed coellident 25 m/tick**5 0.0 REAL default dedaration miss kemc links kemc missile burn speed coellident 27 m/tick**8 0.0 REAL default dedaration miss kemc links kemc missile burn speed coellident 27 m/tick**8 0.0 REAL default dedaration miss kemc links kemc missile burn speed coellident 29 m/tick**8 0.0 REAL default dedaration miss kemc links kemc missile burn speed coellident 29 m/tick**8 0.0 REAL default dedaration miss kemc links kemc missile burn speed coellident 29 m/tick**10 0.0 REAL default dedaration miss kemc links kemc missile burn speed coellident 29 m/tick**10 0.0 REAL default dedaration miss kemc links kemc missile burn speed coellident 20 m/tick**10 0.0 REAL default dedaration miss kemc links kemc missile burn speed coellident 20 m/tick**10 0.0 REAL default dedaration miss kemc links kemc missile burn speed coellident 20 m/tick**10 0.0 REAL default dedaration miss kemc links kemc missile burn speed coellident 20 m/tick**10 0.0 REAL default dedaration miss kemc links kemc missile burn speed coellident 20 m/tick**10 0.0 REAL default dedaration miss kemc links kemc missile burn speed coellident 20 m/tick**10 0.0 REAL default dedaration miss kemc links kem	kem_bum_speed_coeff[ 3]					[miss_kem.c]mlssile_kem_init	[miss_kem.c]missile_kem_lire;	
kem missile burn speed coellident 24 m/tick**5 0.0 REAL default declaration miss. kem.c missile burn speed coellident 25 m/tick**5 0.0 REAL default declaration miss. kem.c missile burn speed coellident 25 m/tick**7 0.0 REAL default declaration miss. kem.c missile burn speed coellident 27 m/tick**3 0.0 REAL default declaration miss. kem.c missile burn speed coellident 28 m/tick**3 0.0 REAL default declaration miss. kem.c missile burn speed coellident 28 m/tick**3 0.0 REAL default declaration miss. kem.c missile burn speed coellident 28 m/tick**3 0.0 REAL default declaration miss. kem.c missile burn speed coellident 29 m/tick**3 0.0 REAL default declaration miss. kem.c missile burn speed coellident 20 m/tick**3 0.0 REAL default declaration miss. kem.c missile burn speed coellident 20 m/tick**3 0.0 REAL default declaration miss. kem.c missile burn speed coellident 20 m/tick**3 0.0 REAL default declaration miss. kem.c missile burn speed coellident 20 m/tick**3 0.0 REAL default declaration miss. kem.c missile burn speed coellident 20 m/tick**3 0.0 REAL default declaration miss. kem.c missile burn speed coellident 20 m/tick**3 0.0 REAL default declaration miss. kem.c missile burn speed coellident 20 m/tick**3 0.0 REAL default declaration miss. kem.c missile burn speed coellident 20 m/tick**3 0.0 REAL default declaration miss. kem.c missile burn speed coellident 20 m/tick**3 0.0 REAL default declaration miss. kem.c missile burn speed coellident 20 m/tick**3 0.0 REAL default declaration miss. kem.c missile burn speed coellident 20 m/tick**3 0.0 REAL default declaration miss. kem.c missile burn speed coellident 20 m/tick**3 0.0 REAL default declaration miss. kem.c missile burn speed coellident 20 m/tick**3 0.0 Missile burn speed coellident 20 missile burn speed coellident 20 missile burn speed coellident 20 missile burn speed coelliden							( i.e. t leniarile from init-	simpet/data/ms km bs.d
kem missile burn speed coetlident as m/tick**6 0.0 REAL default dedaration miss kem.cjmlastle kem_intt insisile burn speed coetlident as m/tick**6 0.0 REAL default dedaration miss kem.cjmlastle kem_intt insisile burn speed coetlident as m/tick**3 0.0 REAL default dedaration miss kem.c insisile burn speed coetlident as m/tick**3 0.0 REAL default dedaration miss kem.c insisile burn speed coetlident as m/tick**3 0.0 REAL default dedaration miss kem.c insisile burn speed coetlident as m/tick**3 0.0 REAL default dedaration miss kem.c insisile burn speed coetlident as m/tick**3 0.0 REAL default dedaration miss kem.c insisile burn speed coetlident as m/tick**10 0.0 REAL default dedaration miss kem.c insisile burn speed coetlident as m/tick**10 0.0 REAL default dedaration miss kem.c insisile burn speed coetlident as m/tick**10 0.0 REAL default dedaration miss kem.c insisile burn speed coetlident as m/tick**10 0.0 REAL default dedaration miss kem.c insisile burn speed coetlident as m/tick**10 0.0 REAL default dedaration miss kem.c		in inital home eneed coefficient 24	m/tick*5	0.0	REAL	default declaration miss_kem.c	muse Kem. Climastie Kein, 11 u.c.	
kem missile burn speed coeflicient 25 m/tick**6 0.0 REAL detault declaration miss, kem.c finis,	kem_bum_speed_coeff[ 4]	Kem missue pariti streat company				[miss_kem.c]missile_kem_init	imiss kemejmissie kem iire,	
kem nissile burn speed coeflicient as m/tick**6 00 REAL default declaration miss kem.c linis kem.c lin							IMISS Kem Citingshire Kem III	bad with the hand
kem missile barn spred coefficient as m/tick*7 0.0 REAL default declaration miss kem.c linisa_kem.clmdselle_kem_init linisa_kem.c linisa_kem.clmdselle_kem_init		as the property of the same as	m/lick**6	0.0	REAL	default declaration miss_kem.c	miss kem.c/missile_kem_inut	מוווחופו/ חשוש/ היווי
kem missile barn speed coeflident a6 in/tick**7 0.00 REAL initsa_kem.clmisale_kem_inits kem.c initsa_kem.clmisale_kem_inits kem.clmisale_kem_initsale barn speed coeflident a9 in/tick**8 0.00 REAL default dedaration miss_kem.c initsa_kem.clmisale_kem_initsale_barn speed coeflident a9 in/tick**9 0.00 REAL default dedaration miss_kem.c initsa_kem.clmisale_kem_initsale_barn speed coeflident a9 in/tick**10 0.00 REAL default dedaration miss_kem.c initsa_kem.clmisale_kem_initsale_barn speed coeflident a9 in/tick**10 0.00 REAL default dedaration miss_kem.c	kem_bum_speed_coeff[ 5]	Kem missue Dam speed Comment = 5				[miss_kem.c]missile_kem_init	miss kem cjimissile Kem III'e;	
kem missile barn spreed coeflident 26 in/tick**7 0.0 REAL default declaration miss kem.c Inits, kem.c missile barn spreed coeflident 27 m/tick**3 0.0 REAL default declaration miss kem.c limits, kem.c missile barn spreed coeflident 28 m/tick**9 0.0 REAL default declaration miss kem.c limits kem.c limits kem.c missile barn spreed coeflident 28 m/tick**9 0.0 REAL default declaration miss kem.c limits kem.c missile barn spreed coeflident 29 m/tick**10 0.0 REAL default declaration miss kem.c missile barn spreed coeflident 29 m/tick**10 0.0 REAL default declaration miss kem.c missile barn spreed coeflident 29 m/tick**10 0.0 REAL default declaration miss kem.c missile barn spreed coeflident 29 m/tick**10 0.0 REAL default declaration miss kem.c missile barn spreed coeflident 29 m/tick**10 0.0 REAL default declaration miss kem.c missile barn spreed coeflident 29 m/tick**10 0.0 REAL default declaration miss kem.c missile barn spreed coeflident 20 m/tick**10 0.0 REAL default declaration miss kem.c missile barn spreed coeflident 20 m/tick**10 0.0 REAL default declaration miss kem.c missile barn spreed coeflident 20 m/tick**10 0.0 REAL default declaration miss kem.c missile barn spreed coeflident 20 m/tick**10 0.0 REAL default declaration miss kem.c missile barn spreed coeflident 20 m/tick**10 0.0 REAL default declaration miss kem.c missile barn spreed coeflident 20 m/tick**10 0.0 REAL default declaration miss kem.c missile barn spreed coeflident 20 m/tick***10 0.0 REAL default declaration miss kem.c missile barn spreed coeflident 20 m/tick***10 0.0 REAL default declaration miss kem.c missile barn spreed coeflident 20 missile barn spreed coeflident 20 m/tick***10 0.0 REAL default declaration miss kem.c missile barn spreed coeflident 20 missile barn spreed c							miss kem.clmissile kem ily	
kem missile burn speed coellident ag m/tick*** 00 REAL default declaration missile burn speed coellident ag m/tick*** 00 REAL default declaration missile burn speed coellident ag m/tick*** 00 REAL default declaration missile burn speed coellident ag m/tick**** 00 REAL default declaration missile burn speed coellident ag m/tick***** 00 REAL default declaration missile burn speed coellident ag m/tick***** 00 REAL default declaration missile burn speed coellident ag m/tick***** 00 REAL default declaration missile burn speed coellident ag m/tick****** 00 REAL default declaration missile burn speed coellident ag m/tick************************************			11.4.00	00	KI:AL	default declaration miss_kem.c	[miss_kem.c]missile_kem_init;	sinuret/data/ms_km_DS.d
kem missile burn speed coellident 27 m/tick**\$ 0.0 REAL default dedaration miss.kem.c linise_kem.cjmiselle_kem_Init kem missile burn speed coellident as m/tick**9 0.0 REAL default dedaration miss.kem.c linise_kem.clmiselle_kem_Init kem missile burn speed coellident as m/tick**10 0.0 REAL default dedaration miss.kem.c linise_kem_clmiselle_kem_Init kem missile burn speed coellident as m/tick**10 0.0 REAL default dedaration miss.kem.c	kam burn speed_coeff[ 6]	kem missile burn speed coemidem 46	· · · · · · · · · · · · · · · · · · ·	;		Imiss kem.clmuselle_kem_init	Inles kem.c]missile_kem_fire;	
kem missile burn speed coefficient ag m/tick**8 00 REAL default declaration miss.kem.c imissile burn speed coefficient ag m/tick**9 00 REAL default declaration miss.kem.c imissile burn speed coefficient ag m/tick**10 00 REAL default declaration miss.kem.c imiss.kem.c imissile burn speed coefficient ag m/tick**10 00 REAL default declaration miss.kem.c							miss kem.cimissile kem lly	
ken missile burn speed coeliident ag m/tick**9 0.0 REAL default declaration miss kenn.c missile burn speed coeliident ag m/tick**10 0.0 REAL default declaration miss kenn.c missile burn speed coeliident ag m/tick**10 0.0 REAL default declaration miss kenn.c missile burn speed coeliident ag m/tick**10 0.0 REAL default declaration miss kenn.c			000/7/1/	00	REAL	default declaration miss, kem.c	miss_kem.c/missile_kem_int;	simnet/data/ms_km_bs.d
kem missile burn speed covilident ag m/tick**9 0.0 REAL default declaration miss kem.c  kem missile burn speed coellident ag m/tick**10 0.0 REAL default declaration miss kem.c	kem_bum_speed_coeff[ 7]	kem missile burn speed coefficient ay	· ·	è		imiss kem.cjmissile_kem_init	[miss_kem.c]missile_kem_fire;	
kem missile burn speed coellident ag m/tick**9 0.0 KEAL default dedaration miss kem.c miss_kem.c miss_kem.c miss_kem.c miss_kem.c missile burn speed coellident ag m/tick**10 0.0 KEAL default dedaration miss_kem.c							iniss kem.cimissile kem ily	
kem missile burn speed coellident ag m/tick**10 0.0 NEAL default declaration missile burn speed coellident ag m/tick**10 0.0 NEAL ints.kem.clmissile.kem_int			9.7	00	REAL	default declaration miss kem.c	miss_kem.c]missile_kem_init;	simnet/data/ms_km_be.d
kem missile burn speed coellident 29 m/tick**10 0.0 KEAL default declaration miss kem.c	kem bum speed coeff[ 8]		m/ack_y	3		fmiss kem.clmissile kem init	[miss_kem.c]misslle_kem_fire;	
kem missile burn speed coellident ag m/tick**10 00 KEAL delault dedaration miss kem.c							(miss kem.c)missile kem fly	
kem missile burn speed coelildent ag in ruc. 19			0100 1011/10	00	KEAL	default declaration miss kem.c	miss kem chissile kem init;	simnel/data/ms_km_bs.d
	kem_bum_speed_coeff[ 9]	kem missile burn speed coemident ag	2			[miss_kem.c]missile_kem_init	[miss kem.c]missile_kem_ine; [miss kem.c]missile_kem_fly	

NOTE 1 one tick is equal to one frame or 1/15th of a second

## TABLE 5.1.45. - KEM MISSILE COAST SPEED COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of	DEFAULT	DATA	CSU WHERE SET OR	CSU WHERE USED	DATA SOURCE
		MEASURE (NOIR 1)	VALUE	TYPE (NOTE 2)	CALQUICATED		
kem_coast_speed_coeff[ 0]	kein nilssile coast speed coefficient ag	m/tick	105.52162	REAL	delault declaration miss kem.c	Indes kem chuksile kem Init;	simnet/data/ms_km_cs.d
					inuss kem chinesile kem init	lilling retit child solie Activity	
kem_coast_speed_coeff[ 1]	kern missile coast speed coefficient at	m/tick*2	-1.0157285	REAL	default declaration miss_kem.c	miss kem.clnussile kein init;	summet/data/ms_km_cs.d
	-			_	Imiss_kem.cjmissile_kem_init	iniss kem.cimisalle kem ily	
kam coast speed coeff[ 2]	kem missile coast speed coefficient as	m/tick*3	5.6124330e-3	KEAL	delault declaration miss kem.c	[miss_kem.c]missile_kem_init;	simnet/data/ms_km_cs.d
					(miss kem.c)missile kem init	fmiss kem climissile kem fly	
kam coast soeed coaff[ 3]	kem missile coast speed coefficient as	m/tick**4	-1.6262608e-5	REAL	default declaration miss_kem.c	[miss_kem.c]missile_kem_init;	simnet/data/ms_km_cs.d
	_				Inlsa kem.chuksile kem inli	finiss kein chnisalle kein fly	
kem coast speed coeff[ 4]	kem missile coast speed coefficient as	m/lick**5	1.8991932e-8	REAL	default declaration miss_kem.c	Imise_kem.clnussile_kem_init;	sinuset/data/ms_km_cs.d
					(miss_kem.c]missile_kem_Init	[miss_kem.c]missile_kem_fly	
kem coast speed coeff[ 5]	kein missile coast speed coefficient ac	m/tick**6	0.0	REAL	default declaration miss_kem.c	Indes kem.cjmissile_kem_init;	simnet/data/ms_km_os.d
					Imiss kem.cjinissile kem init	Inlas kem.cjmirsile kem fly	
ben cost speed cost! 61	kem missile coast speed coefficient ac	m/tick*7	0.0	REAL	default declaration miss_kem.c	Iniss_kem.c]missile_kem_init;	simnet/data/ms_km_cs.d
					(miss_kem.c]missile_kem_init	(miss kem.c)missile kem fly	
Very coast speed coaff[ 7]	kem missile coast speed coefficient av	m/tick*8	0.0	KEAL	default declaration miss_kem.c	[miss_kem.c]missile_kem_init;	simmet/data/ms_km_cs.d
					[miss_kem.c]missile_kem_init	(miss_kem.clmissile_kem_fly	
Vem court speed coaff B1	kem missile roast sneed coefficient as	m/llck*9	0.0	KEAL	default declaration miss_kein.c	Imise kem.cjmissile kem init;	simnet/data/ms_km_cs.d
					[miss_kem.clinissile_kem_init	Imiss kem.c]missile kem fly	
them comes comed (9)	ben missile roast stood coefficient as	m/ück**10	0.0	KEAL	default declaration miss kein.c	[miss_kem.c]missile_kem_init;	sinuet/data/ms_km_cs.d
					[miss kem.c]missile_kem_init	Indes kemelmissile kem fly	

HOTE 1 one little equal to one frame or 1/15th of a second NOTE 2 REAL is a "C" mean DEFME for type flowi.

- 413 -

## TABLE 5.1.46. - KEM MISSILE BURN TURN COEFFICIENT DATA ARRAY

NAME OF DATA ELEMENT	DESCRIPTION	UNITS OF MEASURE	DEFAULT	DATA	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
		MOTE 1	LECTOR	1011 2 1011 2	delanth declaration mice born	indes born chalecile born full:	simmet /data/ms km bt d
rem_burn_tum_coeff[ U]	kein nussiie cosiie oi maximum tum   during burn coefficient ag	You / balkon	200000	J. Carlot	Inles ken.clmissile kein init	[miss_kem_fmissile_kem_fly	
kem bum tum coeff[ 1]	ken missile cosine of maximum turn	costrad)/tick*2	-6.2386917e-7	REAL	default declaration miss_kem.c	imiss_kein.chnissile_kein_init;	slinnet/data/ms_km_bt.d
	during burn coefficient as				Imiss kem chrissile kem init	[miss_kem.c]missile_kem_fly	
kem_bum_tum_coeff[ 2]	kem missile cosine of maximum turn	cos(rad)/lick**3	1.6146426e-7	REAL	default declaration miss_kem.c	[miss_kem.c]missile_kem_init;	simmet/data/ms_km_bt.d
	during burn coefficient az				Imiss_kem.chnisstle_kem_init	[miss_kem.c]missile_kem_fly	
kem bum tum coeff[ 3]	kem missile costue of maximum turn	con(rad)/dck**4	-9.720142e-7	KEAL	default declaration nuiss kein.c	Indsa kein chnissile kein init;	simmet/data/ms_km_bt.d
	during burn coefficient as				linisa kem.clinissile kem init	[miss_kem.c]missile_kem_fly	
kem burn turn coeff[ 4]	kern missile cosine of maximum turn	cos(rad)/lick**5	0.0	REAL	default declaration miss kem.c	[miss_kem.c]missile_kem_init;	simnet/data/ms_km_bt.d
	during burn coefficient 24				[miss_kem.c]missile_kem_Inlt	[miss_kem.c]missile_kem_fly	
ken bun tum coefff 51	kem intestle costne of maximum turn	cos(rad)/tick*6	0.0	REAL	delault declaration miss_kern.c	Indss_kem.c]inissile_kem_init;	simmet/data/ins_kin_bt.d
	during burn coefficient as				(miss kem.chnissile kem init	[miss_kem.c]missile_kem_fly	
kem bum tum coeff[ 6]	kem missile cosine of maximum turn	cos(rad)/tick*7	0.0	REAL	default declaration miss_kem.c	Imiss_kem.cjmissile_kem_init;	simmet/data/ms_km_bt.d
	during burn coefficient ag				(miss_kem.c)missile_kem_init	(miss_kein.c)missile_kein_fly	
kem bum tum coeff[ 7]	kem missile cosine of maximum turn	cos(rad)/lick**8	0.0	REAL	default declaration miss_kem.c	[miss_kem.chnissile_kem_init;	simnet/data/ms_km_bl.d
	during burn coefficient a7				[miss_kem.c]missile_kem_init	Indes_kem.c/missile_kem_fly	
kem bum tum coeff[ 8]	kem missile cosine of maximum turn	cos(rad)/lick**9	0.0	REAL	default declaration miss_kem.c	Indes kem.chilesile kem Init;	slinnet/data/ms_km_bt.d
	during burn coefficient ag				(miss_kem.c)missile_kem_init	(ndss_kem.c)missile_kem_fly	
kem bum tum coeff[ 9]	kem missile cosine of maximum turn	cos(rad)/lick*10	0.0	REAL	default declaration miss_kem.c	[miss_kem.chnissile_kem_init;	sinunet/data/ms_km_bt.d
	during burn coefficient ag				[miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_fly	

NOTE 1 one tek n equit to one frame or 1/15th of a second NOTE 2 REAL is a "C" mad a DEFAE for type float.

## TABLE 5.1.47. - KEM MISSILE COAST TURN COEFFICIENT DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of	DEFAULT	DATA	CSU WHERE SET OR	CSU WHERE USED	DATA SOURCE
		MEASURE [NOTE 1]	VALUE	TYPE [MOIE 2]	CARCULATED		
kem_cosst_tum_coeff[ 0]	kem missile cosine of maximum turn during coast coefficient ao	cos(rad)/Eck	0.99753111	REAL	default declaration miss_kem.c [miss_kem.chrissile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	slinnet/data/ms_km_ct.d
kem_coast_tum_coeff[ 1]	kem missile cosine of maximum turn during coast coefficient as	cos(rad)/tick**2	5.5817986e-5	REAL	default declaration miss_kem.c [miss_kem.c]mlssile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_ct.d
kem_coast_tum_coeff[ 2]	kem missile cosine of maximum turn during coast coefficient a2	cos(rad)/tick**3 -5.1276276e-7	-5.1276276e-7	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_ct.d
kem_coast_tum_coaff[ 3]	kem missile cosine of maximum turn during coast coefficient ag	cos(rad)/lick**4 2.2388593e-9	2.2388593e-9	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	sinnet/data/ms_km_ct.d
kem_coast_tum_coeff[ 4]	kem missile cosine of maximum tum during coast coefficient a4	cos(rad)/tick**5 -5.1964622€-12	-5.1964622€-12	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_ily	sinnet/data/ms_km_ct.d
kem_coast_tum_coeff[ 5]	kem missile cosine of maximum turn during coast coefficient as	cos(rad)/tlck**6 4.5499104e-15	4.5499104e-15	REAL	default declaration miss_kem.c [rniss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_ct.d
kem_coast_tum_coeff[ 6]	kem missile cosine of maximum turn during coast coefficient as	cos(rad)/tick*7	0.0	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simnet/data/ms_km_ct.d
kem_coast_tum_coeff[ 7]	kem missile cosine of maximum turn during coast coefficient a?	cos(rad)/fick**8	0.0	KEVL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	Iniss_kem.c]missile_kem_init;  miss_kem.c]missile_kem_fly	sinwet/data/ms_kin_d.d
kem_coast_tum_coeff[ 8]	kein missile cosine of maximum tum during coast coefficient ag	cos(rad)/lick**9	0.0	REAL	default declaration miss_kem.c [miss_kem.c]missile_kem_init	[miss_kem.c]missile_kem_inlt; [miss_kem.c]missile_kem_fly	simnel/data/ms_km_ct.d
kem_coast_tum_coaff[ 9]	ken missile cosine of maximum turn during coast coefficient ag	cos(rad)/tick**10	0:0	REAL	default declaration niss_kem.c [niss_kem.c]nissile_kem_init	[miss_kem.c]missile_kem_init; [miss_kem.c]missile_kem_fly	simmet/dala/ms_km_ct.d

NOTE 1 one lick is equal to one frame or 1/15th of a secon

#### TABLE 5.1.48. - NLOS MISSILE CHARACTERISTICS DATA ARRAY

NAME OF DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT	DATA TYPE INOIE 2)	CSU WHERE SET OR CALCULATED	CSU WHERE USED	DATA SOURCE
nlos_miss_cher[ 0]	NLOS_LOCK_THRESHOLD;		0.953153895	REAL	default declaration miss_mlos.c; finiss_mlos.c/missile_mlos_init	[miss_nlos.c]missile_nlos_fly	simiet/data/ms_nl_ch.d
nlos_miss_char[ 1]	NLOS_MAX_1URN_ANGLIS	radians/licks	0.03490659	REAL	default declaration miss_nlos.c; [miss_nlos.c]	Imiss_nlos.clnussile_nlos_ily	simnet/data/ms_nl_ch.d
nlos_miss_char[ 2]	NLOS_VERTICAL_FLIGHT_TIME;	licks	48.0	REAL	detault declaration miss_nlos.c; [miss_nlos.c]missle nlos.clmissle nlos init	Inies_ntos.chnissile_ntos_init	sinuvet/data/ms_nl_drd
rlos_miss_char[ 3]	NLOS_DECLINE_FLIGHT_TIME,	ticks	105.0	REAL	detault declaration miss_nlos.c. imiss_nlos.cjmissile_nlos_init	[miss_nlos.c]missile_nlos_init; [iniss_nlos.c]inissile_nlos_ detectibility	simnei/data/ms_nl_ch.d
nlos_miss_char[ 4]	NLOS_LEVEL_ILIGHT_TIME;	ücks	140.0	REAL	default declaration miss_nlos.c; [miss_nlos.c]miselle_nlos_init	[miss_nlos.c]missile_nlos_ detectibility	sinnet/data/ms_nl_dr.d
nlos_miss_char( 5)	NEOS_ARM_TIME; alos missile arm time	ticks	20.0	REAL	default declaration mise_nlos.c; [miss nlos.c]missile_nlos_thit	[miss_nlos.cjmissile_nlos_fly	simnel/data/ms_nl_ch.d
nlos_miss_char[ 6]	NICOS BUNCOUT TIME; time of powered	ticks	22.5	REAL	default declaration miss_nlos.c; [miss_nlos.c]missile_nlos_init	iniss_nlos.cjmissile_nlos_fly	simnet/data/ms_nl_ch.d
nios_miss_char[ 7]		licks	8000.0	REAL	default declaration miss_mlos.c; [miss_mlos.c]missile_mlos_mit	[inlss_nlos.c inissile_nlos_ily	sinnet/data/ms_nl_ch.d
nlos_miss_char[ 8]	SPEED_0;		mmm:11	KEAL	default declaration rulss_rulos.c; [miss_rulos.c;	[miss_nlos.c]missile_nlos_fly	simuet/data/ms_nl_ch.d
nlos_miss_char[ 9]	SPEED_1;		§3333333	KEAL	default declaration miss_nlos.c; [mlss_nlos.c]missile_nlos_lnit	Imiss_nios.c]ndssile_nios_ity	sinmet/data/ms_nl_ch.d
nlos_miss_char[10]	1HETA_0;		0.013962634	REAL	default declaration miss_nlos.c; [miss_nlos.c]missle_nlos_init	[miss_nlos.c]missile_nlos_ily	simnet/data/ms_nl_drd
nlos_miss_char[11]	SIN_UNGUIDE; sine of level tilght [4.0		0.069756474	REAL	delault declaration miss_nlos.c; [miss_nlox.c]missile_nlos_init	[miss_nlos.c]missile_nlos_lly	sinuset/data/ms_nl_dt.d
nlos_miss_char[12]	COS_UNCUIDE; cosine of level flight [4.0 degrees pitch] for an unguided ruos missile		0.997564050	REAL	default declaration miss_nlos.c; [miss_nlos.c]missile_nlos_init	miss_nlos.cjnussile_nios_ily	simnel/data/ms_ni_ch.d
nlos_miss_char[13]	SIN CLIMB; sine of the delta pitch angle [3.5 degrees] for a climbing nos missile		0.004072424	REAL	default declaration miss_mlos.c [miss_nlos.c]missile_nlos_init	[miss_nios.cjnussile_nos_iiy	binuer, data/me of chd
nlos_miss_char[14]	COS CLIMB; cosine of the delta jaich angle		0.999991708	REAL	default declaration miss_mos.c; [miss_nlos.c]missile_nlos_init	(miss_nios.cjinussie_inos_ii)	papariani/asis/damis
nlos_miss_char[15]	SIN_LOCK; sine of the lock cone angle [9.0 degrees] for a locked-on nlos missile		0.156434465	REAL	default declaration miss_nlos.c [miss_nlos.c]missile_nlos_init	(miss_nios.cjmssie_nios_ii)	billuret/ data/ me_m_card
nlos_miss_char[16]	COS LOCK; cosine of the lock cone angle [9.0]		0.987688341	REAL	default declaration miss_nlos.c; [miss_nlos.c]misslie_nlos_init	[miss_nlos.c]missile_nlos_ily	פווותובו/ משוד/ וווים ווים ביות
nlos_miss_char[17]	COS TERM; cosine of the territoral angle [0.0		0.984307753	REAL	default declaration miss_nlos.c; [miss_nlos.c]missile_nlos_init	(miss_nlos.c mlssile_nlos_lly	simnet/data/ms_nl_dn.d
nlos_miss_char[18]	GOS_LOSE; rosine of the angle [20.0 degrees]		0.939692621	REAL	default declaration miss_nlos.c; [miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_fly	simnet/data/ms_nl_ch.d
nlos_miss_char[19]	NOT USED		0:0	REAL	default declaration miss_nlos.c [miss_nlos.c]missile_nlos_init	(miss_nlos.c]missile_nlos_lly	6imnet/data/ms_ni_dr.d

NOTE 1 one tick is equal to one frame or 1/15th of a secon NOTE 2 REAL is a "C" made DEFRIK for type float.

#### TABLE 5.1.49. - NLOS MISSILE POLYNOMIAL DEGREE DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of	DEFAULT	DATA	CSU WHERE SET OR	CSU WHERE USED	DATA SOURCE
		MEASURE	VALUE	TYPE (MOIF 1)	CALCHICATED		
nlos_miss_poly_deg[ 0]	NLOS_BUKN_SPEED_DEG; polynomial		detault: 1	ini	delault declaration miss_nlos.c;	(miss_nlos.c)missile_nlos_	simmet/data/ms_nd_bs.d
	degree for 140s nuissile burn speed coefficient		range: 0109		(miss_nioe.c)missite_nioe_	int; Imica plos climicalle plos	
	data affay					(he:	
						Indee plos chuissile plos fly	
plos miss poly deal 11	NLOS COAST SPEED DEG; polynomial		delault: 3	Int	default declaration miss_nlos.c;	[miss_nlos.c]missile_nlos_	simmet/data/ms_nl_cs.d
	degree for no missile coast speed coefficient		range: 0 to 9		[nulss_nlos.c]missile_nlos_	Init;	
	data array		:		tot	(infra_nlos.c]missile_nlos_fly	
nlos miss poly deal 2]			0	Int	default declaration miss plos.c		
nlos miss noly deof 31			0	Juj	default declaration miss_nlos.c		
nlos_miss_poly_deg[ 4]			0	int	default declaration miss plos.c		

IE 1 min a "C" type for integer.

## TABLE 5.1.50. - NLOS MISSILE BURN SPEED COEFFICIENT DATA ARRAY

NAME of DATA SIEMENT	DESCRIPTION	UNITS of	DEFAULT	DATA	CSU WHERE SET OR	CSU WHERE USED	DATA SOURCE
		MEASURE	VALUE	TYPE INOTE 23	CALCULATED		
10 1000 1000 1000	alos missile burn speed coefficient ac	m/lick	0.0030333	KEAL	default declaration miss_nlos.c	(miss_nlos.c)missile_nlos_init;	sinnet/data/ins_nd_bs.d
In Triangle Property of	default is 67.0 m/sec				[miss_nlos.chnissile_nlos_init	(miss_nlos.chnissile_nlos_fire;	
						(miss nios chrissile nios liy	
also been seed coeff 1]	plos missile burn speed coefficient ar:	m/tick*2	1.25777777	REAL	default declaration miss_nlos.c	miss_nlos.chnissile_nlos_init;	sumnet/data/ms_N_DS.d
	Company on Chilton 170 110 110 100				[miss_nlos.c]missile_nlos_init	[mise_nlos.c]missile_nlos_fire;	
						(miss nlos.c)missile nlos fly	
10 207	alex missile hum greed coefficient as	m/lick*3	0.0	REAL	default declaration miss plos.c	[miss_nlos.chnissile_nlos_init;	simnet/data/ms_nl_bs.d
"Tipon"speds"Coau 5	7				[miss_nlos.c]missile_nlos_init	[miss_nios.chnissile_nios_fire;	
						[miss_nlos.chnissile_nlos_fly	
10. 377	ales miselle hurn enand conflicient as	m/11ck**4	0.0	REAL	default declaration miss nlos.c	[miss_nlos.c]missile_nlos_init;	simnet/data/ms_nl_bs.d
los parm speed coerr 3)	וותפ וותפפונה מתיון בלהברת הביווניהיים				(miss_nios.c)missile_nios_init	[miss_nlos.c]missile_nlos_fire;	-
						(miss_nlos.c)missile_nlos_fly	
17 337	nice missile hum enced coefficient as	m/lick*5	0.0	REAL	default declaration miss_nlos.c	[mise_nlos.chnissile_nlos_init;	simnet/data/ms_nl_bs.d
Nios_bum_speed_coeff	to make the second or the seco				[miss_nlos.c]missile_nlos_init	[miss_nlos.c]missile_nlos_fire;	
						(miss_nlos.c)missile_nlos_fly	

2015 1 one lick is equal to one frame or 1/15th of a second

## TABLE 5.1.51. - NLOS MISSILE COAST SPEED COEFFICIENT DATA ARRAY

coellident a tr. feet feet feet feet feet feet feet fee	MEASURE [NOIF 1] m/tick	VALUE	ě	411111111111		
nlos missile const speed coellident ag. default is 327.2858074 m/sec nlos missile const speed coellident ag. default is -21.460954 m/sec*2 nlos missile const speed coellident ag. default is 0.8227650 m/sec*3 nlos missile const speed coellident ag.	m/tick		10 IC	CARCULATED		
default is 327.2858074 m/sec hos missile coast speed coellident a; default is -21,460954 m/cec*2 nlos missile coast speed coellident a; default is 0.8227650 m/sec*3 nlos missile coast speed coellident a;		30.46972319	KEAL	default declaration miss_nlos.c	[inise_nlos.c]missile_nlos_linit;	sinunet/data/ms_ul_cs.d
nios nuissile cosat spreed coelident a 1; default is -21,460954 nu/sec*2 nios misile cosat spreed coelident a 2; default is 0.8272650 nu/sec*3 nios misile cosat spreed coelident a 3;				(miss_nios.cjmissile_nios_init	[mise_nios.chnisslie_nios_lly	
default is -21.46095/4 m/sec**2 nlos missile coast speed coefficient a2; default is 0.8227650 m/sec**3 nlos missile coast speed coefficient a 3;	m/tick*2	m/tick*2 -9.7721160e-2	REAL	default declaration miss_nlos.c	[miss_nlos.c]missile_nlos_init;	einmet/data/ms_nl_cs.d
nlos missile coast speed coeflicient a2; default is 0.8227660 m/sec*3 nlos missile coast speed coeflicient a3;				(miss_nlos.c)missile_nlos_init	[miss_nlos.c]ndssile_nlos_fly	
	m/lidk*3	m/lid**3   1.2433925e-4	REAL	default declaration miss_nlos.c	(miss_nlos.c)missile_nlos_init;	simmet/data/ms_nl_cs.d
nlos missile coast speed				[miss_nlos.chnissile_nlos_inft	Imiss_nlos.c/mlssile_nlos_fly	
/ (ACT. 10.0 1.1 1.1.	m/lick**4	m/lick**4 -5.4061501e-8	REAL	default declaration miss_nlos.c	[miss_nlos.chnissile_nlos_indt;	simmet/data/ms_nl_cs.d
default 18 -0.0155200 ml/sec. 4				imiss nios chulseile nios init	Imiss_nlos.c]missile_nlos_fly	
ning coast sneed coeff[ 4] nlos missile coast speed coefficient a4	m/tick**5	0.0	REAL	default declaration miss_nlos.c	[miss_nlos.chnissile_nlos_init;	slinnet/data/ms_nl_cs.d
				(miss_nlos.chvissile_nlos_init	(miss_nlos.c)missile_nlos_fly	

NOTE 1 one tick is equal to one frame or 1/15th of a signal to the form of 1/15th of a signal to the forty of flows.

- 418 -

#### TABLE 5.1.52 - HYDRA ROCKET CONFIGURATION DATA ARRAY

MANUEL CONTACTORNEY	DESCRIPTION	UNITS of	DEFAULT	DATA	CSU WHERE SET OR	CSU WHERE USED	DATA SOURCE
NAME OF UNIA ELEMENT		MEASURE INOTE 1)	VALLIE	TYPE [NOTE 2]	CALCULATED		
hydra_rkt_char[ 0]	hydra launcher postiion, X	E	4.5	REAL	default declaration rwa_hydra.c; [rwa_hydra.c]hydra_init	[rwa_hydra.c]hydra_indt;	/sinnet/data/rwa_hydr.d
hydra_rkt_char[ 1]	hydra launcher positon, Y	Ē	0.5	REAL	default declaration rwa_hydra.c; [rwa_hydra.c]hydra_init	Irwa_hydra.clhydra_init;	/sinnet/data/rwa_hydr.d
hydra_rkt_char[ 2]	hydra launcher postion, Z	E	-20	REAL	default declaration rwa_hydra.c; [rwa_hydra.c]hydra_lnit	[rwa_hydra.c]hydra_init	/sinnet/data/rwa_hydr.d
hydra_rkt_char[ 3]	nilis of Soviet articulation	in the	0.101	REAL	default declaration rwa_hydra.c;	trwa_hydra.cJhydra_init	/sinnet/data/rwa_hydr.d
hydra_rkt_char[ 4]	degrees of hull negative pitch	deg	-5.0	REAL	default declaration rwa_hydra.c; [rwa_hydra.c]hydra_init	frwa_hydra.c hydra_inlt; frwa_hydra.c hydra_sel_pylon_ articulation	/simnet/data/rwa_hydr.d
hydra_rkt_char[ 5]	degress of maximum articulation	8ap	19.0	REAL	default declaration rwa_hydra.c	[rwa_hydra.c]hydra_init	/simnet/data/rwa_hydr.d
hydra_rkt_char[ 6]	degrees of minumum articulation	deg	-15.0	REAL	default declaration rwa_hydra.c; [rwa_hydra.c]hydra_init	[rwa_hydra.c hydra_init	/slmnet/data/rwa_hydr.d

NOTE 1 one tick is equal to one frame or 1/15th of a second NOTE 2 REAL is a "C" mad o DEPME for type float.

- 419 -

#### TABLE 5.1.53 - HYDRA ROCKET CHARACTERISTICS DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE [NOIR 1]	DEFAULT	DATA TYPE (NOTE 2)	CSU WHERE SET OR CALCULATED	CSJ WHERE USED	DATA SOURCE
rkt_hydra_char[ 0]	NISI_BURST_SPREAD; twin bursts which are 3 meters apart	E	1.5	KEAL	detault dedar Mon rkt_ydra.c; [rkt_hydra.c missile_hydra_init	irkt_hydra.chuissile_hydra_iruit; [rkt_hydra:chuissile_hydra_set_ pyjon_articulation	/simmet/data/rkt_hydr.d
rkt_hydra_char[ 1]	M261_BURST_HEIGHT; release submunitions 180 feet	æ	54.864	REAL	default declaration rkt_hydra.c; [rkt_hydra.c]miselle_hydra_init	[rkt_hydra.c]missile_hydra_init; [rkt_hydra.c]missile_hydra_fire	/simnet/data/rkt_hydr.d
rkt_hydra_char[ 2]	M261_BURSI_RANGE; 0 meters in front of	E	0:0	REAL	default declaration rkt_hydra.c; [rkt_hydra.c]missile_hydra_init	[rkt_hydra.c]missile_hydra_init	/simnet/data/rkt_hydr.d
rkt_hydra_char[ 3]	M261_BURST_SPREAD; twin bursts are 13 motors apart	E	6.0	REAL	default declaration rkt_hydra.c. [rkt_hydra.c]missile_hydra_init	[rki_hydra.cjmusaile_hydra_init	/snmet/data/fkt_nydr.d
rkt_hydra_char[ 4]	M255_BUKS1_KANCE; release datts 150 meters in front of larget	ш	0.021	REAL	default declaration rkt_hydra.c; [rkt_hydra.c]missile_hydra_init	{rkt_hydra.c]missile_hydra_lnit	/simnel/data/rki_hydr.d
rkt_hydra_char[ 5]	M255_BURST_SPREAD; twin bursts are 35 meters apart	ш	16.0	REAL	default dedaration rkt_hydra.c [rkt_hydra.c]missile_hydra_init	[rkt_hydra.c]mlssile_hydra_init	/simnet/dala/rkt_hydr.d
rkt_hydra_char[ 6]	FLECH 60 MAX RANGE; darts fly a total of 750 meters	ш	750.0	REAL	default declaration rkt_hydra.c; [rkt_hydra.c]missile_hydra_init	[rkt_hydra.c]missile_hydra_init	/wmmet/data/rkt_nydr.d
rkt_hydra_char[ 7]	hydra mlnimum range	E	90.0	REAL	default declaration rkt_hydra.c; [rkt_hydra.c]miselle_hydra_init	irkt_hydra.c/missile_hydra_init; irkt_hydra.c/missile_hydra_set_ pyion_articulation	/simnet/data/rkt_hydr.d
rkt_hydra_char[ 8]	hydra maximum range for Soviet S-5 57mm rocket	E	5000.0	REAL	default dedaration rkt_hydra.c; frkt_hydra.c missile_hydra_init	[rkt_hydra.c missile_hydra_init;	/simnet/data/rkt_hydr.d
rkt_hydrs_char[ 9]	hydra maximum rango tor M151	u .	7000.0	KEAL	delault dedaration rkt_hydra.c; [rkt_hydra.c]missile_hydra_init	[rkt_hydra.chnissile_hydra_inut; [rkt_hydra.chnissile_hydra_fire; [rkt_hydra.chnissile_hydra_set_ pylon_articulation	/eimet/data/tkt_hydr.d
rkt_hydra_char[10]	hydra maximum rango tor M261	ē	7000.0	REAL	delault dedaration tkt. hydra.c; [rkt_hydra.c]missile_hydra_init	[rkt_hydra.chudssile_hydra_init; [rkt_hydra.c]mdssile_hydra_fire; [rkt_hydra.chudssile_hydra_set_ pylon_articulation	/simnet/data/rkt_hydr.d
rkt_hydra_char[11]	hydra maximum range lor M255	E	3200.0	REAL	default declaration tkt_hydra_c; [rkt_hydra_c]missile_hydra_init	[rkt_hydra.c]missile_hydra_init; [rkt_hydra.c]missile_hydra_fire; [rkt_hydra.c]missile_hydra_set_ pylon_articulation	/simnet/data/rkt_hydr.d

NOTE 1 one tick is equal to one frame or 1/15th a NOTE 2 REAL is a "C" mado DEFRE for type float

## TABLE 5.1.54. - SUBMUNITIONS M73 CHARACTERISTICS DATA ARRAY

						Canada transfer	DATA COLIDOR
NAME of DATA ELEMENT	DESCRIPTION	UNITS of MEASURE	DEFAULT	DATA	CSU WHERE SET OR CALCULATED	CSU WHERE USED	
		MUIL	-	1	1 6 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	feuch m74 climiterile m74 init:	/simpet/data/sub m73.d
10 Park 01	75% of gravity - (75% * (9.8m/sec*2)/225   m/s*2/lick*2 0.03266667	m/c. 2/110-7	_	KEAL	derault declaration but 111/3.5	'and anomaly and	
to lieus c / w cons	11-1-12				(sub m73.c)missile_m73_init	(sub m/3.c)missile m/3 drop	
	OCKS 2/			1.32	J. Carrie deal seation such m73 c.	lenb m73 clmissile m73 init:	/simmet/data/sub m73.d
1 1.44	Includition (all with + /- 8.8 degrees	deg	126	KEAL	detault derdalation but in och		
fi ligury W. a Cuarl 17	11-1-1-1-1	:			fsub m73.clmissile m73 inft	[sub_m73.c]missile_m/3_get_impact	
	angular displacement					faut m73 clmissile m73 init.	/simnet/data/sub m73.d
	Instruction (all with + /. 12 15 decrees	deg	22.7	KEAL	detault declaration sub_my.c.	dan Triboniforchin and	The state of the s
Sub_M/3_char_ 4	ואומיוובוובו ביו ווווו ול ידים בישר היווו				faut m73 climicalle m73 int	fash m73 climissile m73 get impact	
	anoular displacement				Sup litt octingsite title and		
	0						

NOTE 1 one tick is equal to one frame or 1/15 th of a secon NOTE 2 REAL is a "C" macro DEFINE for type float.

## TABLE 5.1.55. - SUBMUNITIONS FLECHETTE CHARACTERISTICS DATA

NAME OF DATA ELEMENT	DESCRIPTION	JO STINI	DEFAULT	PATA	CSU WHERE SET OR	CSU WHERE USED	DATA SOURCE
		MEASURE	AVECUE	(NOTE 1)			
sub_flech_char[ 0]	maximum speed < 100	Z.,W	10000.0	REAL	default declaration sub_flech.c	[sub_flech.c)missile_flechette_init;	/sinnet/data/sub_flec.d
•	•				(sub_flech.c]missile_flechette_Init	[sub_flech.c]missile_flechette_fly	
sub_flech_char[ 1]	Hechettes lly in a cylinder with a radius of 17.5 m*2	Z.,III	306.253	REAL	default declaration sub_flech.c;	(sub_flech.c)missile_flechette_init;	/sinnet/data/sub_llec.d
	meters and a length of 750 meters				[sub_flech.c]missile_flechette_init	[sub_flech.c]missile_flechette_fly_	
sub_flech_char[ 2]	FLECH 60 MAX RANCE; darts fly a total of	E	750.0	REAL	default declaration sub_flech.c;	[sub_flech.c]missile_flechette_init;	/sinnet/data/sub_ilec.d
•	750 ineters				[sub flech.c]missile flechette init	(sub flech.c)missile flechette fly	

El mina "C" type for inleger.

#### TABLE 5.1.56. - FLECHETTE SPEED DATA ARRAY

NAME of DATA ELEMENT	DESCRIPTION		DEFAULT	DATA	CSU WHERE SET OR	CSU WHERE USED	DATA SOURCE
		MEASURE [NOTE 1]	VALUE	TYPE (MOTE 2)	CALCULATED		
flechatte_speed_coef[ 0]	Hechette speed coellident ag	m/tick	41.75	REAL	default declaration sub_flech.c	[sub_flech.c]missile_flechette_init;	/simmet/data/flec_spd.d
	•				[sub flech.c]missile flechette init	(sub flech.c)missile flechette fly	
(lechatte speed coeff 1)	l flechette speed coefficient as	m/tick/m	-0.20397254	REAL	default declaration sub_flech.c	[sub_flech.chrissile_flechette_fnlt;	/sinmet/data/flec_spd.d
	-				[sub_flech.c]missile_flechette_init	(sub_flech.c)missile_flechette_fly	
flechatte sneed coeff 21	Hechelle speed coefficient as	m/lick/m°2	m/tick/m**2 0.00022724278 REAL	REAL	default declaration sub_flech.c	[sub_flech.c]missile_flechette_init;	/simmet/data/llec_spd.d
	•				[sub_flech.c]missile_flechette_init	[sub_flech.c]missile_flechette_fly	
flachatta speed coaff 31	flechette speed coefficient an	m/tick/m"3	m/tick/m*3 -0.00000008633 REAL	REAL	default declaration sub_flech.c	[sub_flech.c]missile_flechette_init;	/sinmet/data/flec_spd.d
					(sub_flech.c)missile_flechette_init	(sub_flech.c)missile_flechette_fly	
flachatta soand confl 41	flechette speed coefficient a	m/tlck/m**	0.0	REAL	default declaration sub_flech.c	[sub_flech.chrissile_flechette_init;	/simnet/data/llec_spd.d
	•				[sub_flech.c]missile_flechette_init	(sub flech.c)missile flechette fly	

NOTE 1 ove tick is equal to one frame or 1/15th o NOTE 2 REAL is a "C" snacro DEFINE for type float. 6. Notes.

NONE.

#### Appendix A - RWA AirNet Call Tree Structure.

The following appendix contains information for convenience in document maintenance and understanding of the overall CSCI architecture. This call tree is not all inclusive, i.e., it only contains the calls from the top-level down to the CSU of interest in this document. Other CSCs and CSUs have been included in the Call Tree for clarity and reference.

read\_pars.c read\_pars.c read\_pars.c

> eye\_to\_screen\_distance vconfig\_file1 vconfig\_file2

rwa\_main.c rwa\_main.c rwa\_main.c rwa\_main.c

												amed					
ammo_map_file sdamage_file devices file	calib_file	het_calib_file	unesn_me asid_map_file	idle_filter_file sim filter file	priority_list_file	register_file	subsystems	atoi	cig_set_number_subsystems	default db version	db_override	cig_use_database_override_named	ded_override	set_ded_name	overlay_file	waypoint_list	constants_file

read\_pars.c read\_pars.c read\_pars.c

read\_pars.c read\_pars.c

read\_pars.c read\_pars.c read\_pars.c read\_pars.c read\_pars.c

read\_pars.c

read\_pars.c read\_pars.c

read\_pars.c

read\_pars.c read\_pars.c

read\_pars.c

read\_pars.c read\_pars.c read\_pars.c

read\_pars.c

ntework\_dont\_really\_open\_up\_ethernet set\_assymetric\_on need\_to\_fill\_initial\_munitions set\_request\_receive\_size set\_request\_send\_size use\_static\_debug force\_other f\_velocity debug sscanf printf

cig\_not\_using\_graphics

fclose

RWA AIRNET CALL TREE STRUCTURE	8th
AIRNET (	7th
RWA	6th
	ع

	rwa_main.c	rwa_main.c rwa_main.c					•	rwa_main.c	•	rwa_keybrd.c	rwa_keybrd.c						rwa_mann.c	7	read_pais.c	read_pars.c	read_pais.c	read_pais.c	o diem com	I Wa_manic	Julios ema	rwa sound.c			o diem ewir	TWa_Mann	rwa main c	1 W a_11111111		rwa_tads.c			rwa tads.c
																-																					
		;																																÷			
8th																																					
6th 7th																																					
5th 6				levice																										,	ed						
4th				network_get_network_device		le		rver	ď.			S							a	name	ion	rersion				•	pun	•	device		cig_use_database_override_named			10.1	<u>S</u>		mind
3rd	e		•	rk_get_1		ker_enab	skb	oility_se	Init Setu	ılly_use	, 	er_effect.	re_dir	series		_mode	su		db_nam	default_db_name	db_versi	ult_db_\		tion	110 <sup>-8</sup> n	nse	dont_use_sound	•	retwork_		ase_over		•	se_mode_	21 VISIUIL. 7	يو ي	,
2nd	guise_override	guise_to_use print help	printf	netwo		head_eye_tracker_enable	use_print_checkb	use_intervisibility_server	Intervisibility Init Setup	keyboard_really_use	use_keyboard	het_enable_laser_effects	het_set_damage_dir	het_set_laser_series	y_if	v_pkt_verbose_mode	print_overruns		get_default_db_name	defaı	get_default_db_version	defaı	lata	initial_activation	rva_turn_debug_on	sound_dont_use	dont	print	network_set_network_device	db_subsys	se_datab	ded_subsys	set_ded_name	terrain_verbose_mode_on	tads_set_intervisionity stripn	nrintf	
1st	guise_	guise_to_us print help	i		exit	head_e	use pr	use_ir	Interv	keybo	use_k	het_en	het_se	het_se.	set_mi	v_vkt	print	strcpy	get_d	ı	get_d		movedata	initial	rva_tı	sonne			netwo	db_sı	cig_u	qeq_;	set_de	terrai	rads		

rwa\_status.c

rwa\_pots.c

rwa\_pots.c rwa\_pots.c

pil\_cyc\_roll\_I pil\_cyc\_roll\_c

fopen printf exit

rwa\_sound.c rwa\_status.c rwa\_status.c

RWA AIRNET CALL TREE STRUCTURE d 3rd 4th 5th 6th 7th 8th	strcpy dev mask	sim_state_startup sim_state simulate_state_machine	sim_state initial_bbd bbd_init	print) dtad_init mem assign_shared_memory	ser_heartbeat_init idc_init	dont_use_sound sounds	sound_reset dont_use_sound	sounds fifo_init	Sound_error fprintf fflush	veh_sound_array	status_preset idc_values	equipment_status status_out	timers_init	fopen
	cpy	startup n_state state_mae	m_state itial_bbd d_init	ınıı) ad_init en assiyı	r_heartber c_init nund_init	do So So	So			lini sute	st.	oo Ys	imers_inil	of E
1st 2nd	cis cis	sim_state_startup sim_state sim_state simulate_state_m	sin in	ž ž	es idi	3				7	Ď.		41 12	<b>L</b>

rwa\_sound.c rwa\_sound.c rwa\_mem.c

main.c main.c main.c

main.c main.c rwa\_sound.c rwa\_sound.c rwa\_mem.c

rwa\_sound.c

rwa\_pots.c

pitch_r   1   1   1   1   1   1   1   1   1   1	strenip pil_cyc_pitch_d pil_cyc_pitch_c	rwa_pots.c rwa_pots.c
iil_pedal_I iil_pedal_r iil_pedal_r iil_pedal_r iil_coll_r vols_dreck_two pg_trav_I pg_trav_I pg_trav_r pg_trav_r pg_trav_r po_trav_r po_trav_r po_trav_r po_trav_r po_trav_r po_trav_r po_trav_r inti cobjects inti inti inti ggt_net_handle	p::_cy_pica_c pil_cyc_pitch_r	rwa_potsc
iil_podal_c iii_podal_r iii_bodal_r iii_bodal_r iii_coll_d oli_coll_r oli_coll_r pg_trav_I pg_trav_I pg_trav_c pg_trav_i pg_trav_c po_trav_c po_trav_c po_trav_c po_trav_c po_trav_c po_trav_c po_trav_c inti iiit iiit iiit iiit iiit iiit iii	pil_pedal_l	rwa_pots.c
il_pedal_r il_pedal_r il_coll_d il_coll_d il_coll_r obsdreck_two pgtrav_r pgtrav_r pgelev_d pgelev_r ppelev_r potrav_r potrav_r potrav_r poelev_r poelev_d potrav_r int	pil_pedal_c	rwa_pots.c
iii_coll_d iii_coll_d ois_clieck_two pg_trav_I pg_trav_I pg_trav_I pg_trav_I pg_elev_c pg_elev_c po_trav_I po_trav_I po_trav_I po_trav_I po_trav_I po_trav_I po_elev_d po_elev_d po_elev_c po_elev_d po_elev_c po_elev_r init init sget_net_landle	pil_pedal_r	rwa_pots.c
ii_coll_r  out_clerk two  pg_trav_I pg_trav_I pg_trav_r pg_clev_d pg_elev_c pg_elev_r po_trav_I po_trav_I po_trav_r po_clev_d po_trav_r po_clev_d po_elev_c po_elev_c po_elev_c po_elev_c po_elev_c int int int int int int ireally_use_intwork get_net_landle	pil_coll_d	rwa_pots.c
ous_cneck_two pg_trav_I pg_trav_I pg_trav_C pg_trav_C pg_elev_C pg_elev_C po_trav_I po_trav_I po_trav_I po_trav_I po_trav_C po_elev_C po_elev_C int close int int int int int int int get_inet_handle	pil_coll_r	rwa_pots.c
pg_tav_r pg_tav_r pg_tav_r pg_clev_d pg_elev_c pg_elev_r po_trav_l po_trav_l po_trav_r po_elev_d po_elev_d po_elev_r trav int tra	pots_check_two	o ofor car
pg_trav_c pg_trav_r pg_clev_d pg_elev_c pg_elev_r po_trav_I po_trav_I po_trav_r po_clev_d po_clev_c po_clev_c po_clev_c po_clev_c po_clev_r init init init init init set_ireally_use_network get_iret_handle	cpg_uav_1	TWA_DUST.
pg_trav_r pg_clev_d pg_clev_c pg_clev_c pg_clev_r po_trav_r po_trav_r po_trav_r po_clev_d po_clev_c po_clev_c po_clev_r tolosc init tit tit tit tit tit tit tit tit ti	cpg_trav_c	I wa_pous.c
pg_clev_d pg_clev_c pg_elev_r po_telev_r po_trav_l po_trav_r po_trav_r po_clev_d po_clev_c po_clev_c po_clev_r tdose int	cpg_trav_r	rwa_pots.c
pg_elev_c pg_elev_r po_trav_I po_trav_I po_trav_r po_trav_r po_elev_d pp_elev_d po_elev_d po_elev_r close with rouize up trouize up	cpg_elev_d	rwa_pots.c
pg_elev_r po_trav_1 po_trav_1 po_trav_c po_trav_r po_clev_d po_clev_r close init cup pup nup nup init san i_really_use_network get_net_handle	cpg_elev_c	rwa_pots.c
po_trav_I po_trav_c po_trav_c po_trav_r po_elev_d po_elev_c po_elev_r close with with rouize wp init init san_i_really_use_network get_net_handle	cpg_elev_r	rwa_pots.c
po_trav_c po_trav_r po_lev_d po_elev_d po_elev_r po_elev_r int close int ronize up ronize up init init get_net_liandle	cpo_trav_I	rwa_pots.c
po_trav_r po_elev_d po_elev_c po_elev_r closeup rouize up init init init	cpo_trav_c	rwa_pots.c
po_elev_d po_elev_r po_elev_r po_elev_r close init re up ronize up init init get_net_handle	cpo_trav_r	rwa_pots.c
po_elev_c po_elev_r close initobjects re try try init init init get_net_handle	cpo_elev_d	rwa_pots.c
po_elev_r close init objects re up ronize up init init san_i_really_use_network	cpo_elev_c	rwa_pots.c
close init - objects re up ronize iup init can_i_really_use_network get_net_handle	cpo_elev_r	rwa_pots.c
nit reobjects re np ronize inp init init get_net_handle	fclose	
up ronize up init init can_i_really_use_network	network_init	
re inp ronize inp init init init get_net_handle	obj_create_objects	
up ronize up init can_i_really_use get_net_handle	cig_prepare	
ronize init san_i_really_use get_net_handle	buffer_setup	
init init can_i_really_use get_net_handle	ynchronize	
init can_i_really_use get_net_handle	startup	
can_i_really_use get_net_handle	repair_uninit	
can_i_really_use get_net_handle	init	
_can_i_really_use _get_net_handle it	top	
ork_get_net_handle init	_can_i_really_use	
init	ork_get_net_handle	
	filter_init	

TREE STRUCTURE	
CALL TREE ST	844
RWA AIRNET CALL	7+1
RWA	444

	read_pars.c read_pars.c	main.c	main.c	rwa_main.c		main.c main.c	read_pars.c	read_pars.c	read_pars.c	reau_pais.c	read_pars.c	read_pars.c	read_pars.c	read_pars.c	rwa main.c	rwa_config.c			rwa_config.c		rwa confio	rwa_config.c	o bijaco cim	rwa_config.c	
KWA AIKINEI CALE INEE SINOCIONE	,								•.														•		
CALLI	8th																								
AIKINE	7th								-																
	5th 6th ile				mulator_type	_reconfig_startup cig_startup_func	<i>cig_reconfig_start</i> nfig_file1	vconfig_file1	ap_file	ammo_map_file	file	nap_file	le_read ofile	map_file	file	oit Oit	init_symbol_table	;	<i>reader_find_file</i> data file		tag.	velucle_name munitions table	get_symbol	weapons_info_size weapons_info	۲
	2nd 3rd 4th get_priority_list_file priority_list_file	tp te_idle	printf sim_state	veh_spec_startup rtc_init_clock	printf network_set_sinulator_type	use_cig_reconfig_startup cig_startup_func	cig_reconf get_vconfig_file1	veonf	get_ammo_map_file	amme	map_file_read get veh map	veh_map_file	map_vehicle_file_rea	asid_	map_read_asid_file	init_activ	init_s init_s	printf	<i>reader_fii</i> data file	exit	find_tag	vehic muni	S-128	weat weat	malloc
	2nd get_pric	rva_setup sim_state_idle		veh_sp												•									

sun\_wayed.c sun\_wayed.c rwa\_ammo.c rwa\_config.c rwa\_config.c

rwa\_config\_determine\_ammo\_needed

rwa\_resupply\_in progress

rwa\_resupply\_completed

is\_fuel\_vehicle

véhicle\_force

is\_friendly

rwa\_config.c rwa\_config.c rwa\_config.c

rwa\_config.c rwa\_config.c

rwa\_ammo.c

resupp.c

rwa\_config.c

rwa\_config.c

resupp.c

resupp.c

rwa\_main.c

	·	, ,
3rd 4th 5th 6th 7th 8th resupply_in_progress hungry_for_ammo rwa_config_get_was_munition_index was resupply_get_ammo_offcred ammo_offcred leftwing_stores ammo_type_full anmo_type_full rightwing_stores turret_stores	rwa_config_get_was_munition_index was rwa_config_get_was_position_name was softp_label leftwing_stores rightwing_stores turret_stores mun_set_veh_spec_resupply_completed veh_spec_resupply_completed rwa_resupply_started rwa_resupply_in progress printf	rwa_config_get_was_munition_index was rwa_config_get_was_position_name was softp_label leftwing_stores rightwing_stores turret_stores turret_stores veh_spec_resupply_started veh_spec_resupply_started lunge_files use_intervisibility_server
2nd		

rwa\_config.c rwa\_config.c rwa\_config.c

rwa\_config.c rwa\_config.c

rwa\_config.c

rwa\_config.c rwa\_config.c

RWA AIRNET CALL TREE STRUCTURE

**1st** 

rwa\_config.c rwa\_config.c rwa\_config.c ammo.c

rwa\_config.c

resupp.c resupp.c

rwa_main.c rwa_status.c rwa_status.c rwa_status.c	rwa_mem.c rwa_status.c	rwa_status.c . rwa_mem.c rwa_status.c	rwa_status.c	rwa_status.c
2nd 3rd 4th 5th 6th 7th 8th Intervisibility Init Intervisibility Synchronize timers_simul veh_spec_idle status_simul frame_counter monitor_status idc_values hard_dead	fifo_hard st_com fijo_enqueue softi_dead fijo_softi softo_dead ser_heratbeat ser_dead net_xnnt_failed net_dead	set_xmt_failed dtad_failed dtad_dead sound_dead	st_sound temperature current_lemperature voltage12P current_plus12 HILIMIT-12P plus12_dead	voltage12N current_minus12 HILIMIT_12N minus12_dead LOLIMIT_12N voltage5 current_plus5

		rwa_status.c	rwa_status.c	rwa_status.c	rwa_status.c	rwa status.c	rwa_status.c	rwa_status.c	rwa_status.c	rwa_status.c	rwa_status.c			٠	rwa_main.c	rwa_main.c	rwa config.c	rwa config.c	rwa config.c	ı			rwa_main.c	rwa_network.c	fuelsys.c	fuelsys.c	rwa_config.c	rwa_config.c	rwa_config.c	rwa_config.c	rwa_config.c	ammo.c	ammo.c	3	rwa_conng.c
RWA AIRNET CALL TREE STRUCTURE 2nd 3rd 4th 5th 6th 7th 8th HILIMIT_5	plus5_dead LOLIMIT_5	need_to_set_host_red	cquipment_status	need_to_set_cig_red	need_to_set_hard_red	need_to_set_soft_red	need_to_set_sound_red need_to_set_sound_red	need to set voltage12P red	need to set voltage12N red	need_to_set_voltage5_red	need_to_sct_net_red	status_out	keyboard_simul	io_simul_idle	initial_activation	need_to_fill_initial_munitions	printf	rwa_conng_innanze_inumunons	previous_veincie_type	ממום זווב	Jina_tag	oet sumbol	init activ	fill vehicle status	fuel get current level	fuel_struct	rwa config get was munition type	was	rwa config_get_was_munition_index	was	leftwing_stores	ammo_check_availability	ammo_index_ok	print	rightwing_stores

aerodyn\_init engine\_init

	rwa_engine.c rwa_engine.c		rwa_engine.c rwa_engine.c	0	rwa_engine.c	•	rwa_engine.c	rwa_engine.c	rwa_engine.c	rwa_engine.c	rwa_engine.c	rwa_engine.c		rwa_engine.c	rwa_engine.c	rwa_engine.c		rwa_engine.c		rwa_engine.c	rwa_engine.c	rwa_engine.c	rwa_engine.c	rwa_engine.c	rwa_engine.c	rwa_engine.c	rwa_engine.c		rwa_engine.c	rwa_engine.c	rwa_engine.c		rwa_engine.c	rwa_engine.c	rwa_engine.c	
RWA AIRNET CALL TREE STRUCTURE	minutes_of_flight ine damage engine	controls_start_failure_lamp_flashing	engine_is_damaged	controls_failure_lamp_off	engine_is_damaged	fail_init_failure	engine_break_engine	engine_status	engine_speed	number_of_engines	engine repair engine	engine_repair_engine_oil	controls_failure_lamp_off	engine_is_damaged	engine_status	number_of_engines	engine_damage_transmission_filter	engine_repair_transmission_filter	controls_failure_lamp_off	transmission_is_damaged	engine_break_transmission	engine_break_engine	engine_status	engine_speed	number_of_engines	engine repair_transmission	engine_repair_transmission_filter	controls_failure_lamp_off	transmission_is_damaged	engine_repair_engine	engine_repair_engine_oil	controls_failure_lamp_off	engine_is_damaged	engine_status	number_of_engines	
117	ın.																																			0
, .	nic									-																										
7 2	7III7																																			

controls\_failure\_lamp\_off

engine\_repair\_engine\_oil

engine\_repair\_engine

engine\_speed number\_of\_engines engine\_is\_damaged

engine\_damage\_transmission\_filter

number\_of\_engines

engine\_status

engine\_repair\_transmission\_filter

controls\_failure\_lamp\_off transmission\_is\_damaged

engine\_break\_engine

engine\_break\_transmission

7.6	411	1 1	6th 7th 8th	
ora	#111	me	11	000000000000000000000000000000000000000
			engine_power	I wa_ciigiiic.c
			engine_percent_torque	rwa_engine.c
			engine speed	rwa_engine.c
			number of engines	rwa_engine.c
			engine status	rwa_engine.c
			engine is damaged	rwa_engine.c
			transmission_is_damaged	rwa_engine.c
			starting engine	rwa_engine.c
			integrator gain	rwa_engine.c
			gov p gain	rwa_engine.c
			gov i gain	rwa_engine.c
			last per cent shaft_torque	rwa_engine.c
			last percent torque	rwa_engine.c
			hours of flight	rwa_engine.c
			minutes of flight	rwa_engine.c
,			old minutes of flight	rwa_engine.c
			engine_damage_engine_oil	rwa_engine.c
			controls_start_failure_lamp_flashing	
			engine_is_damaged	
			engine_repair_engine_oil	
			controls_failure_lamp_off	
			engine_is_damaged	
			fail_init_failure	
			engine_break_engine	
			engine_status	

RWA AIRNET CALL TREE STRUCTURE

2nd

**1st** 

- A-14 -

	rwa_ rwa_ rwa_	!	rwa	rwa_ rwa_	Reference # W003092; 31 March 1995 Volume 1 of 3; Rev. 0.	3 .0
		rwa_engine.c	rwa_engine.c	rwa_engine.c	rwa_config.c rwa_aerodyn.c rwa_aerodyn.c rwa_aerodyn.c rwa_config.c	rwa_config.c rwa_aerodyn.c
RWA AIRNET CALL TREE STRUCTURE	2nd 3rd 4th 5th 6th 7th 8th engine_status engine_speed number_of_engines	engine_repair_transmission_filter engine_repair_transmission_filter controls_failure_lamp_off	transmission_is_damaged engine_repair_engine engine_repair_engine_oil	controis_raiture_iainp_oit engine_is_damaged engine_status number_of_engines	cos vec_init ground_force vehicle_mass_init ground_init find_cubic_func fprintf get_constants_file aerodyn_read_simple_constants fopen printf fgets strtok strcmp sscanf fclose rwa_config_get_front_support front_support aero_body_point_set_front_wheels body_point_set_front_wheels printf pr	rwa_coming_get_rear_support rear_support aero_body_point_set_rear_wheel

3rd	4th	5th	RWA 6th	AIRNE 7th	T CALI	RWA AIRNET CALL TREE STRUCTURE 6th 7th 8th	23	•	d Constant
		body_	body_point printf					rwa	rwa_ground.h
alt_init	nit					•			
gunmnt_	gunmnt_init								
-ches	sad uninit								
SAD	•	PORT							
sad_init	init								
amm	ammo_configuration_menu_init	uration	_menu	'_init					,
weat	weapons_init							rwa_	rwa_weapons.c
•	gunm	gunmnt_element	nent					rwa_	rwa_weapons.c
	111111	:							
	rotate_	rotate_init_element	ment						0.000
	cig_set_	et_proc_	_proc_hit_msg	38					Iwa_cig.c
	,	rwa_	process	rwa_process_msg_hit_return	vit_ret	ırı			rwa_cig.c
			f2d_	f2d_vec_copy	*				
			map	map_is_missile	ile			•	:
			mis	sile_util	comi	missile_util_comm_intersected_poly		in .	util_comm.c
				mis	missile_comm	mmo		nt	util_comm.c
			Ssu	get_veh	1_id_fr	msg_get_veh_id_from_cig_id		,	
			mis	sile_util	com	missile_util_comm_intersected_model			util_comm.c
				mis	missile_comm	nmc		Ħ	util_comin.c
			pro	proc_hit_debug	gnqa				rwa_cig.c
			printf	ıtf					
			mat	map_is_bomb	<u>۽</u>			•	000000
			veh	veh_kinematics	atics	,		S	san stabs.c
			kine	kinematics_range_squared	range_	squared		,	0 04:40
			ma	p_get_n	etwor	map_get_network_type_from_ammo_entry	ntry	νo	sun_stubs.c
			net	network_ifire_init_burst	e_init_	burst			
			neta	work_ifir	re_sent	network_ifire_send_detonation			
			net	work_ifir	re_sena	network_ifire_send_indirect_fire			
			imp	impacts_queue_effect	ffa_ona	ct			
			IW	a_config	get	rwa_config_get_was_munition_index			
			ron	rounds_interp_rounds	terp_r	spunc			
			cig	impact	fron	cig_impact_from_round_fired			
		chan	ge_proc	change process_msg_hit_function	_hit_fi	nction			-
	roun	rounds_init_volley	volley					M.	rwa_rounds.c
		volle	volley_list					M.	iwa_iounds.c
		volle	volley_free	<b>.</b>				<b>*</b>	I Wa_louilds.c

rkt\_hydra.c

rkt\_hydra.c

miss\_hellfr.c

rkt\_hydra.c

miss\_hellfr.c

util\_ball.c

missile\_util\_load\_ball\_traj\_file

ball\_table table\_size

printf

rkt\_hydra.c rkt\_hydra.c

rkt\_hydra.c rkt\_hydra.c rkt\_hydra.c rkt\_hydra.c

rkt\_hydra.c rkt\_hydra.c

rwa\_hydra.c rwa\_hydra.c rwa\_hydra.c rwa\_hydra.c rwa\_hydra.c rkt\_hydra.c

rwa\_hydra.c

rwa\_hydra.c

fuze\_prox.c

ball_load.c	rkt_hydra.c sub_flech.c sun_stubs.c fuze_prox.c fuze_prox.c fuze_prox.c fuze_prox.c	rwa_weapons.c miss_tow.c miss_hellfr.c miss_hellfr.c miss_tow.c rwa_weapons.c	miss_hellfr.c miss_hellfr.c miss_hellfr.c rwa_weapons.c miss_stinger.c miss_stinger.c miss_stinger.c miss_hellfr.c miss_hellfr.c miss_hellfr.c miss_hellfr.c miss_hellfr.c miss_hellfr.c miss_hellfr.c miss_hellfr.c fuze_prox.c fuze_prox.c fuze_prox.c fuze_prox.c
fprintf exit cof getc ungetc fscanf fclose printf	printi flechette_veh_list flechette_is_valid_veh rva_create_output_list missile_fuze_prox_init prox_list prox_free free ptr	r limi squz	missile_hellfire_init speed_factor max_range_limit max_range_limit max_range_squared hellfire_ammo_type stingers mum_stingers stinger_array speed_factor max_range_limit max_range_limit max_range_squared stinger_ammo_type missile_fuze_prox_init prox_list prox_free free_ptr

5th

4th

3rd

2nd

util\_comm.c util\_comm.c

missile\_util\_comm\_init

missile\_comm

			L. G. T.	
			RWA AIRNET CALL TREE STRUCTURE	
2nd	3rd	4th	5th 6th 7th 8th	
		Weapo	weapons config missile	rwa_weapons.c
		-	rwa config get was munition info	rwa_config.c
			, and a second s	rwa_config.c
			missile hellfire set ammo type	miss_hellfr.c
			hellfire ammo type	miss_hellfr.c
			missile hellfire set max range limit	miss_hellfr.c
			max range limit	miss_hellfr.c
			max range squared	miss_hellfr.c
			missile hellfire set speed factor	miss_hellfr.c
			speed factor	miss_hellfr.c
			missile stinger set ammo type	miss_stinger.c
			stinger ammo type	miss_stinger.c
			missile stinger set max_range_limit	miss_stinger.c
			max range limit	miss_hellfr.c
			max range squared	miss_hellfr.c
			missile stinger set speed factor	miss_stinger.c
			speed factor	miss_hellfr.c
			missile tow set ammo type	miss_tow.c
			tow ammo type	miss_tow.c
			missile tow set max range limit	miss_tow.c
			max range limit	miss_hellfr.c
			max range squared	miss_hellfr.c
			missile tow set sneed factor	miss_tow.c
			speed factor	miss_hellfr.c
			nrintf	
			missile util init	util_init.c
			***************************************	ufil comm.c

**1st** 

> controls\_start\_failure\_lamp\_flashing network\_missiles\_init controls\_failure\_lamp\_off weapons\_break\_gun\_major weapons\_repair\_gun\_major weapons\_repair\_hellfire weapons\_break\_hellfire weapons\_break\_stinger weapons\_break\_gun weapons\_repair\_gun fail\_init\_failure

rwa\_weapons.c

rwa\_weapons.c

rwa\_weapons.c

RWA AIRNET CALL TREE STRUCTURE

eth

4th

3rd

2nd

RWA AIRNET CALL TREE STRUCTURE	
CVIL.	8th
AIRNET	7th
RWA	6th

	main.c main.c	rwa_cig.c	main.c	main.c			rwa_main.c rwa_status.c rwa_status.c	1	rwa_sound.c rwa_sound.c rwa_ctl_fsm.c	rwa_ctl_fsm.c rwa_ctl_fsm.c rwa_ctl_fsm.c rwa_ctl_sim.c	
st 2nd 3rd 4th 5th 6th 7th 8th	obj_init_objects cig_startup_func cig_startup_func_FPTR	buffer_reset cig_spec_init cig_nnsg_prepend_request_laser_range	fail_init sim_state_simulate	printf sim_state RTC_FRAME	rtc_start_time RTC_FRAME_GAP bbd_bit_out RTC_TIMERS_SIMUL	rtc_stop_time RTC_FAIL_SIMUL fail_simul	KIC_VEH_STEC_Simulate veh_spec_simulate status_simul	frame_counter  monitor  keyboard_simul  waypoint_editor	sound_simul sound_error	controls_status controls_status controls_failure_val controls_sim_routines controls_pil_cyc_roll_check controls_pil_cyc_pitch_check controls_pil_pedal_check	controls_pil_coll_check controls_copil_trav_check

		rwa_ctl_fsm.c	rwa_ammo.c ammo.c ammo.c
6th 7th 8th controls_copil_elev_check controls_pil_trigger_1_check controls_pil_trigger_1_check controls_cpg_trigger_1_check controls_cpg_trigger_2_check controls_cpg_trigger_2_check controls_cpg_cont_laser_burst_check controls_weapons_master_check controls_weapons_master_check controls_weapons_master_check controls_weapons_cpg_check controls_view_slew_check controls_pil_was_check controls_cpg_was_check controls_cpg_was_check	controls_cpo_auto_track_check controls_slave_check controls_slave_check controls_hover_hold_check controls_wide_fov_check controls_narrow_fov_check controls_narrow_fov_check controls_cpo_sensor_check controls_polarity_check controls_polarity_check controls_radar_warning_flash_check controls_failure_lamp_flash_check controls_failure_lamp_flash_check controls_master_caution_check	controls_manual_range_check controls_failure_edge	simul ammo_quantity_has_changed ammo_indicators_require_updating
5th			~ ~ 5
4th		fprintf nprintf	view_simul ammo_simul amm
<b>3rd</b>			view amm
2nd		•	

RWA AIRNET CALL TREE STRUCTURE

			R	RWA AIRNET CALL TREE STRUCTURE		
pu	3rd	4th	5th 6	6th 7th 8th		
		amme	ammo_check_availability	vailability		
		rightv	rightwing_stores	S. S.	rwa_config.c	J
		turret	turret_stores	,	rwa_config.c	J
		meter	meter_missile1_set	_set		
		meter	meter_missile2_set	set		
		meter	meter_rocket_set	et		
		meter	meter_ammo_set	et		
		resup	ply_receive	resupply_receive_gating_conditions_ok		
		rva_li	rva_lists_off		rwa_ammo.c	ب
		rwa_6	ammo_resu	rwa_ammo_rcsupply_list_id	rwa_ammo.c	ب
		rva b	rva build list			
		rwa 1	rwa fuel resupply list_id	ply_list_id	rwa_ammo.c	ر.
		rva g	rva get output list	list	sun_stubs.c	ب
		rwa_(	config_dete	rwa_config_determine_ammo_needed		
		mnu	set_ammo_	mun_set_ammo_resupply_list		
		unu	set_fuel_re	mun_set_fuel_resupply_list		
		roa dont	out build list	· Isil		
		resupply	ply_simul	I		
	fuel simul	imul	 			
	meter simul	simul			rwa_meter.c	ب
	resunt	olv sin	nul			
	tid bid	hhd hit out				
		RTC RIMA SIMI	MIII			
	7 7 7	Carrie transcript				
	115_314	11 - 111110			rwa simul.c	ب
	rwa_sımuı	imui	•		i :	?
		get_s	get_selected_model	lodel		,
		aerodyn <u>.</u>	lyn_simul		rwa_aerodyn.c	ب
			get_airc	get_aircraft_kinematic_state		
			-	orientation_calc		
			, -	parameters_calc	,	
				true_airspeed	rwa_aerodyn.c	ပ္
				kinematics get true airspeed	rwa_kinemat.c	ပ္
				true airspeed	rwa_aerodyn.c	<u>ر</u>
				altitude	rwa_aerodyn.c	<u>ن</u>
				kinematics get altitude	rwa_kinemat.c	r.c
				altitude	rwa_aerodyn.c	<u>.</u> ر
				angular velocity_vector	rwa_aerodyn.c	 
				kinmatics_get_angular_velocity_vector	rwa_kinemat.c	ن د
				ang_vel	rwa_kinemat.c	o.

1011110112cd_rcioti	• •
kinematics_get_normalized_velocity_vector	rwa_kinemat.c
true_airspeed	rwa_kinemat.c
norm_vel	rwa_kinemat.c
pos unit vel	rwa_kinemat.c
neg unit vel	rwa_kinemat.c
velocity vector	rwa_aerodyn.c_
kinematics get linear_velocity_vector	rwa_kinemat.c
velocity vector	rwa_aerodyn.c
gravity dir vector	rwa_aerodyn.c
kinematics get gravity_vector	rwa_kinemat.c
gravity	rwa_kinemat.c
angle_of_attack	rwa_aerodyn.c
kinematics_get_aoa	•
side_slip_angle	rwa_aerodyn.c
kinematics_get_yaw	,
velocity_to_body	rwa_aerodyn.c
kinematics get velocity_to_body	rwa_kinemat.c
velocity to body	rwa_aerodyn.c
o force	rwa_aerodyn.c
kinematics get o force	rwa_kinemat.c
	rwa_aerodyn.c
vertica	rwa_aerodyn.c
kinematics get vertical speed	rwa_kinemat.c
vertical speed	rwa_aerodyn.c
compute flight	rwa_aerodyn.c
	rwa_aerodyn.c
altitude	rwa_aerodyn.c
air_density	٠
ambient_temperature	rwa_aerodyn.c
air_temperature	•
ambient_pressure	rwa_aerodyn.c
air_pressure	· ·
dynamic_pressure	rwa_aerodyn.c
true_airspeed	rwa_aerodyn.c
pitch_rate	rwa_aerodyn.c
angular_velocity_vector	rwa_aerodyn.c
roll_rate	rwa_aerodyn.c
yaw_rate	rwa_aerodyn.c

RWA AIRNET CALL TREE STRUCTURE

normalized\_velocity\_vector

5th

4th

3rd

2nd

**1st** 

-	rwa_aerodyn.c	rwa_aerodyn.c		rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	,	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c
5th	roll	gravity_dir_vector	atan2	pitch	compute_stab_augmentation_gains	hover hold state	hover hold turned on	pitch damping	roll damping	hover aug roll integrator	hover aug pitch integrator	stab aug yaw integrator	stab aug climb integrator	true airspeed	MAX_ATT_CTL_ANGLE	Sol	velocity_vector	limiter	hover aug roll angle	stab aug roll	set roll attitude	attitude control roll integrator	roll	limiter	attitude_control_roll_command	hover ang pitch angle	stab aug pitch	set pitch attitude	attitude control_pitch_integrator	pitch	limiter	attitude_control_pitch_command	angular velocity_vector	stab ang yaw	stab aug climb	controller cyclic roll	cyclic roll	controller cyclic pitch	cyclic_pitch
4th																																							
3rd																																							
2nd																																							

RWA AIRNET CALL TREE STRUCTURE 6th 7th 8th

controller_tail_rotor  pedal controller_collective  collective  collective  compute_rotor_load_torque  controller_collective  tail_rotor_load_torque  controller_tail_rotor  compute_engine_torque  main_rotor_load_torque  main_rotor_load_torque  altitude  engine_simul  engine_load_torque  altitude  engine_power  gov_p_gain  engine_status  integrator_gain  gov_i_gain  fuel_level_empty  fuel_level_empty  fuel_level_empty  fuel_struct  engine_drive_torque  number_of_engines  engine_percent_torque

3rd

2nd

•	rwa_ctl_fsm.c	rwa_ctl_tsm.c	rwa_ctl_fsm.c	rwa_ctl_fsm.c	rwa_meter.c	rwa_meter.c	rwa_meter.c		rwa_meter.c	rwa_ctl_fsm.c	rwa_ctl_fsm.c	rwa_ctl_fsm.c	rwa_ctl_fsm.c	rwa_meter.c	rwa_meter.c		rwa_engine.	rwa_engine.c	rwa_engine.c		rwa_engine.c	rwa_engine.c	rwa_engine.c	rwa_engine.c		rwa_engine.c	rwa_engine.c	•	rwa_engine.c	rwa_aerodyn.c	rwa_engine.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c
3rd 4th 5th 6th 7th 8th	}	controls_status	controls_failure_status	controls_failure_val	conv frac to percent	torque set val	torque oscillation	softp_ins_panel_set	meter rom set	controls power status	controls status	controls failure status	controls failure_val	conv frac to percent	rpm set val	softp_ins_panel_set	hours_of_flight	minutes of flight	old_minutes_of_flight	sfail_event_occurred	engine is damaged	transmission is damaged	engine sound type	last percent shaft speed	sound make cont sound	last percent torque	rotor oscillation	sound_make_arg_sound	engine oscillation	nowertrain percent shaft speed	ongine get roter percent shaft speed	I best shaft speed	Summer to rotor forces and moments	main rotor thrust	nowertrain percent shaft speed	controller collective	tail rotor thrust	controller tail rotor	force_body_main_rotor

2nd

2nd	3rd	4th	5th	6th 7th 8th	Junposoc com
				MAIN_ROTOR_MAST_TILT_SIN	1 Wa actody
				MAIN ROTOR MAST TILT COS	rwa_aerodyn.c
				force body tail rotor	rwa_aerodyn.c
				moment hody main rotor	rwa_aerodyn.c
				controller cyclic nitch	rwa_aerodyn.c
				controller cyclic roll	rwa_aerodyn.c
				continuity in the form	rwa_aerodyn.c
				mann_10101_1014uc_10au	rwa_aerodyn.c
			duioo	compute_lift_drag_coefficiens	rwa_aerodyn.c
				IIII COCIIICIEII Vatau	rwa aerodyn.c
				side_slip_angle	rwa aerodyn.c
				vstab_lift_coefficient	rwa aerodyn.c
				lift_coefficient_virtual_wing	rwa aerodyn c
				true_airspeed	wa_acrodync
				p drag_fit_coeff	I wa_aciouyii.c
				cubic func	
				angle_of_attack	rwa_aerouyn.c
					•
				total incompressible drag coefficient	rwa_aerodyn.c
					rwa_aerodyn.c
			comp	compute_int_drag_iones	rwa_aerodyn.c
				litt_virtual_wing	rwa aerodyn.c
				dynamic_pressure	rwa aerodyn.c
				lift coefficient_virtual_wing	o more of the o
				lift vstab	rwa_aerodyn.c
				lift exofficient vstab	rwa_acrodyn.c
				ווון רטכווורוכוון אפניים ייין קייי	rwa_aerodyn.c
				total_drag	rwa aerodyn.c
				total_incompressible_drag_coenication	rwa aerodyn.c
			com	compute_body_damping_torces_and_montens	rwa aerodyn.c
				moment_body_damping	rwa aerodyn.c
				pitch_damping	rwa aerodyn.c
				ptich_rate	rwa aerodyn.c
				roll_damping	rwa aerodyn.c
				roll_rate	rwa aerodyn c
				yaw damping	I wa_aciouy ii.c
				yaw rafe	rwa_aerodyn.c
				force body damning	rwa_aerodyn.c
				inducting weeter	rwa_aerodyn.c
				Velocity_versor	rwa_aerodyn.c
			tran	Storm Internal wing force	rwa_aerodyn.c
				VIItuai mingirore	rwa_aerodyn.c
				וונר און וחשו" אא זייל	

**1st** 

	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	•	rwa_aerodyn.c	rwa_aerodyn.c		rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	fuelsys.c	fuelsys.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_ground.h		•	rwa_ground.h		rwa_aerodyn.c	rwa_aerodyn.c	sun_wayed.c	rwa_aerodyn.c	rwa_aerodyn.c		rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c
RWA AIRNET C	out out /m out	lift vstab	drag force	total drag	true airspeed	pitch	sin	velocity to body	lift_body_virtual_wing	vec mat mul	lift body vstab	drag body	generate gravity body force	compute gross weight	vehicle mass	fuel get current level	fuel struct	gross weight	gravity force body	gravity dir vector	gross weight	interact with pround	normalized velocity vector	true airsneed	body point	ground force	ground_torque	grnd	ground_interaction	force_ground_effect	main_rotor_thrust	cig_altitude_above_gnd	sum body forces and moments about ac	force_body	vec_init	force_body_main_rotor	bec_uuu lift bodv virtual wing	lift_body_vstab
	4th																																					
•	3rd																																					
•	Znd																																					

	rwa aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	o autorace ema	rwa_acrouyin.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c		rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c		rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c	rwa_aerodyn.c		rwa aerodyn.c		rwa aerodyn.c	and potanina I							rwa_kmemat.c	rwa_aerodyn.c	o limio omi	I Wa_Silliulic		rwa_kinemat.c	LWa_Killelilat.c
RWA AIRNET CALL TREE STRUCTURE	2nd 3rd 4th 5th 6th 7th 8th	drag_body force body damping	gravity_force_body	ground_force	force_ground_effect	loc ac tail rotor cop	force body_tail_rotor	moment_body_tail_rotor	vec_cross_prod	loc ac virtual wing cop	moment body virtual wing	loc ac vstab cop	moment hody vstab	80 08 00	monomit body co	momont body	moment body main rotor	around forms	woment body damping	and to dynamics kinematics	Sella_u_u_niici_commess	Venter incorp	Included in the second	venicle_mass_mit	force_body	vehicle_forces	moment_body	vehicle_torques	veliicle_update	aerodyn_simple_simul	aerodyn_stealth_simul	printf	bbd bit out	kinematics get roll	roll	fabs	current_bank	sound make const sound	kinematics_get_body_pitch	body_pitch

,	,		
2nd	3rd	4th 5th 6th 7th 8th	
		meter_adi_set	
		kinematics_get_heading	•
		•	rwa_tads.c
		tads_element	rwa_tads.c
		Sig	
		sight	
		laser_range	
		get_cmd_heading_state	
		ling	rwa_meter.c
		ling_val	rwa_cig_2d.c
			sun_wayed.c
			•
			sun_stubs.c
		kinematics_get_d_pos	
		world	
		Infl	
		rotate_get_mat	
		vec_mat_mul	
		meter_ball_set	
		of_tick	,
		St.	rwa_cig_2d.c
			rwa_cig_2d.c
		ing	rwa_cig_2d.c
		alt sen bearing val	rwa_cig_2d.c
			va_cig_2d.c
			rwa_cig_2d.c
	rtc sta	rtc stop time	
	tads_simul	imul	
	firect	II .	
	weap	ıul	rwa_weapons.c
	•	ic_gun_simul	rwa_weapons.c
			wa_inecu.c rwa_tads.c
		п	rwa firectl.c
		ווובנוו־מוובינות־ה/ה/הוחו	ı

RWA AIRNET CALL TREE STRUCTURE 6th 7th 8th

rwa\_weapons.c

rwa\_weapons.c

rwa\_weapons.c

rwa\_weapons.crwa\_weapons.c

rwa\_hydra.c rwa\_config.c

rwa\_weapons.c

rwa\_weapons.c

rwa\_config.c rwa\_weapons.c

rwa\_config.c

**1st** 

sth bcs_turr bcs_set_ cig_2d_ new_gu tads rotate_s bias_ve sight bcs_get bcs_get	S S S S S S S S S S S S S S S S S S S	bcs_tu bcs_tu bcs_se gun_o gun_o cig_2d new_g tads rotate bias_v sight bcs_g	RWA AIRNET CALL TREE STRUCTURE	6th 7th 8th nil weapon select state		ballistics computer rwa_weapons.c		rwa_weapons.c	zb rwa_weapons.c	range rwa_	alc se	ballistics_calc_time bal_calc.c	sart	fprintf		ge	rwa_	rwa_tads.c		ľWa		rw	super elevation rwa_hydra.c	M	ctl rocket selected	cpg weapon select state	
	4th		RWA AIRNET CALL T		bcs_turn_computer_off	bes set ballistics computer	super elevation	qx	$\frac{\tilde{z}}{2}$	bcs range	ballistics calc sc	ballistics_calc	sart	fyrintf	gun out of contstraints	cig_2d_set_status_message	new gun firing state	tads	rotate_8et_angle	bias_vector	sight	bcs get_super_elevation	super elevation	gun limits	firectl rocket selected	cog weapon	

gun\_switch
new\_shot
shot\_counter
shot\_interval
last\_shot
ammo\_typc
turret\_stores
ammo\_check\_availability
leftwing\_stores
tracer\_round\_interval
gun\_impacts\_per\_round
weapons\_fire\_round

5th	6th 7th 8th firs a to w mat	rwa_weapons.c
	tire_g_to_w_mat continuous_gun_sounds	rwa_weapons.c
•	gun_impacts_per_round	rwa_weapons.c
	sound_make_const_sound	Stubs. cms
	veh_kinematics	
	kinematics_get_d_pos	
	vec_scale	
	ıl2f_vec_copy	
	event_get_event	J SUOUBEMI
	bias_vector	January Charles
	projectile_drift	wa_weapons:c
	scaled_rand	
	tads_currently_fixed_forward	rwa_tads.c
	firectl gun_selected_by_pilot	rwa_nrecu.c
	gun_munition_data	rwa_weapons.c
	world	
	gunmnt	rwa_weapous.c
	gunmnt_element	rwa_weapons.c
	rotate_get_loc	
	fixed gun	rwa_raus.c
	fixed_gun_element	rwa_tads.c
	rotate_get_mat	0.000
	sight	1 Wa_dus.c
	gummt get sight to world	rwa_gunmnt.c
	bcs computer status	rwa_weapons.c
	hes booted up	rwa_weapons.c
	hes get super elevation	rwa_weapons.c
	mat rot)init2	
	mat_mat_mul	
	may get tracer from annino entry	;
	ballistics_fire_a_round	ball_fire.c
	rounds update last volley	rwa_rounds.c
	last volley	rwa_rounds.c
	network_can_i_really_use_network	o empore 147m
	null_vehicleID	INC. IIYUI AK
	network send shell fire pkt	
ammo_fired	fired	rwa rounds.c
rounds	rounds_get_volley	rwa_weapons.c
contin	continuous_gun_sounds	•

2nd

rwa_weapons.c	rwa_weapons.c rwa_weapons.c	rwa_weapons.c rwa_weapons.c	rwa_cig_2d.c	1	rwa_weapons.c	rwa_weapons.c rwa_cig_2d.c	rwa_weapons.c	rwa_weapons.c		miss_stinger.c	targ_pursuit.c	rwa_weapons.c	miss_tow.c miss_hellfr.c	sun_stubs.c	miss_hellfr.c	miss_atgm.c	util_eval.c miss atgm.c	util_eval.c	miss_atgm.c
RWA AIRNET CALL TREE STRUCTURE  3rd 4th 5th 6th 7th 8th	gun_ning_sand make_sound_const_sound missile_simul	new_reticle_state current_reticle_state	reticle_on sound_make_const_sound cig_2d_set_inner_box	sight hull	rotate_get_mat	stinger_searching current_search_state	cig_2d_set_stinger_location	rva_dout_build_list	sunger_ready rva_build_list	rotate_set_loc	missile_stinger_pre_launch near_get_preferred_veh_near_vector missile_target_pursuit vec_copy	printf	ان	max_range_limit مناب kinematics	kinematics_range_squared	speed_factor	tow_burn_speed_coeff missile_util_eval_poly	toe_burn_turn_coeff missile_util_eval_cos_coeff	missile_util_eval_poly tow_coast_speed_coeff

	miss atgm.c	taro oromotic	J sol vol est	1915 - 19		miss_tow.c					rwa_weapons.c		o allor soim	a: illian_centt	miss_hellfr.c	miss hollfr	111135_1161111.5	ufil_eval.c	miss_hellfr.c	1		;	miss hellfr.c	J shirts and	Suri_support	•	miss_hellfr.c	tare eround.c	1416-610411415	targ_aguir	targ_agm.c	)   											util_flyout.c						
RWA AIRNET CALL TREE STRUCTURE	4th 5th 6th 7th 8th	tow_coast_turn_coeff		missile_target_level_los	missile_util_flyout	missile tow stop	missile missile	IIIISIIII — IIIISIIII — IIIIIIIIIIIIIII	missile_util_comm_cneck_intersection	missile util comm_check_detonate	hallfrae	TIGHTI 63	laser_point	missile hellfire fly		speed_factor	hollfire him speed coeff	alou land the state of	missile_uti_evar_for	hellfire_coast_speed_coen	sart	· · · · · · · · · · · · · · · · · · ·	: : : : : : : : : : : : : : : : : : : :	max_range_limit	yeh kinematics	binanatics range somared	VIII CHIMITE TAN SCOOL	max_range_squared	missile target_ground	ministration of	missie_taigct_abii	agm_scek	sgrt	vec_scale	vec add	dus son	VEC_311	vec_copy	sgrt	vec_dot_prod	vec scale	add second	tion line disci	Illisone and the state of the s	915_390	vec_dot_prod	vec_copy	sart	vec scale
	3rd																																																
	2nd																																																
	1st																																																

	util_comm.c	sun_stubs.c	util_comm.c	util_comm.c	util_comm.c	o Hollfr o	util_comm.c	util_comm.c	rwa cig 2d.c	rwa_firectl.c	rwa_firectl.c	rwa_tirecti.c	rwa firectl.c	rwa_firectl.c		rwa_cig_zu.c rwa_cig_2d.c		o nomina soim	miss stinger.c	miss_stinger.c	miss_stinger.c	miss_sunger.c util_eval.c	miss_stinger.c	
<u> </u>	add sile_	printf veh_kinematics	kinematics_range_squared missile_comm map_get_get_tracer_from_anmo_entry	store_traj_chou network_send_missile_appearance missile_util_comm_stop_missile	print missile_comm network send_non_impact	network_stop_missile_flyout	missile-hellfire_stop missile_util_comm_stop_missile	missile_util_comm_check_intersection	missile_util_comm_check_detonate	cig_2d_check_tof_countdown	firectl_hellfire_selected nil_woapon_select_state	pil was_position	rwa_config_get_was_munition_info	cpg_weapon_select_state	cpg_was_position	is_printing_tof	laser_range	clg_za_set_tot missile hellfire_calc_tof	missile_stinger_fly_missiles	num_stingers	missile stinger fly	stinger_burn_speed_coeff	missile_util_eval_pory stinger_coast_speed_coeff sqrt	

2nd

miss_hellfr.c	miss_hellfr.c targ_ground.c	targ_unguide.c	o spanos com	rwa_rounds.c	rwa_rounds.c	rwa_rounds.c fuze_prox.c		rwa_rounds.c			sun_stubs.c	rwa rotate.c	!	rwa_kinemat.c		rwa_aerodyn.c
RWA AIRNET CALL TREE STRUCTURE  2nd 3rd 4th 5th 6th 7th 8th  cos  near_get_preferred_veh_near_vector  max_range_limit	ven_kinematics_range_squared max_range_squared missile_target_ground missile_target_intercept_pre_burnout	missile_target_intercept missile_target_unguided missile_util_flyout missile_stinger_stop		rounds_check_volleys	last_volley	rounds_free_volley	nee_pu printf	volley_free free	head_eye_tracker_receive_data head_cye_tracker_send_request Lrf Tick	Lif Err String	RTC_KINEMATICS_SIMUL	kinenatics_simul RTC_TURRET_SIMUL	turret_simul rotate_hull_simul	rotate_sinuil veh_spec_kinematics_simul	world hull	rotate_get_loc altitude

sub\_m73.c util\_comm.c

missile\_m73\_drop missile\_util\_comm\_check\_sub\_mun printf

missile\_comm

util\_comm.c

	1	rwa_aerodyn.c	רשמ_מכוטמאזוי.	rwa aerodync.		rwa kinemat.c	rwa aerodyn.c	rwa kinemat.c	1		rwa kinemat.c	rwa aerodyn.c	rwa aerodyn.c		rwa kinemat.c	3	rwa kinemat.c	rwa aerodyn.c	rwa kinemat.c			rwa hydra.c	rkt hydra.c	rkt hydra c	rkt hydra c	ואוידואמו	Jerphylan	ואנדואמויי				sub_m73.c					
RWA AIRNET CALL TREE STRUCTURE	2nd 3rd 4th 5th 6th 7th 8th	velocity	airspeed		indicated_airspeed	air_density	norm_vel	sin_aoa	COS_aoa	sin_yaw		velocity_to_body	ang_vel	vehicle_angular_velocity	rotate_get_mat	gravity	g_force	vertical_speed	vec_dot_prod	velocity_pitch	asin	body_pitch	roll	heading	acos	het_simul	nul	missile_hydra_fly_rockets		hydra fly	missile_hydra_fly	ball_table	missile_m73_init	missile_flechette_init	printf	missile_hydra_fly:missile_hydra_stop	missile m/3 drop

	sun_stubs.c	sun_wayed.c	sub_m73.c sub_m73.c util_comm.c	sun_stubs.c util_comm.c	sun_wayed.c sub_m73.c	sub_m73.c	util_comm.c util_comm.c sun_stubs.c	sun_wayed.c
RWA AIRNET CALL TREE STRUCTURE	1st 2nd 3rd 4th 5th 6th 7th 8th veh_kinematics kinematics_range_squared	network_ifire_init_burst network_ifire_send_detonation map_get_ammo_entry_from_network_type inpacts_queue_effect network_send_vehicle_inpact	stated_initial  sqrt  traj_up  zero_velocity  missile_util_comm_release_sub_munition	printf veh_kinematics kinematics_range_squared missile_comm store_traj_chord	vec_copy  d2f_vec_copy  map_gct_ammo_entry_from_network_type  network_send_projectile_fire_pkt  impacts_queue_effect  missile_m73_gct_impact	scaled_rand sin vec_scale vec_add missile_m73_impact	missile_util_comm_check_sub_mun  printf  missile_comm  veh_kinematics	network_ifire_init_burst network_ifire_send_detonation map_get_ammo_entry_from_network_type impacts_queue_effect

	sub_m73.c util_comm.c	util_comm.c sun_stubs.c	sun_wayed.c	sub_m73.c sub_m73.c util_comm.c	sun_stubs.c util_comm.c	sun_wayed.c	sub_m73.c	sub_mt3.c util_comm.c	sun_stubs.c util_comm.c
RWA AIRNET C	5th 6tn 7th 8th nctwork_send_vehicle_impact missile_m73_drop missile_util_comm_check_sub_mun	missile_comm veh_kinematics kinematics_range_squared network_ifire_init_burst	u_p;	sqrt traj_up zero_velocity missile_util_comm_release_sub_munition	printf veh_kinematics kinematics_range_squared missile_comm store_traj_clord	event_get_eventu vec_copy d2f_vec_copy map_get_ammo_entry_from_network_type network_send_projectile_fire_pkt	impacts_queue_effect missile_m73_get_impact scaled_rand sin	Ξ	printf veh_kinematics kinematics_range_squared missile_comm

4th

3rd

2nd

o bowew and	sun_waycu.c	rkt_hydra.c sub_flech.c	sub_flech.c util_eval.c	I	rwa_hydra.c	fuze_prox.c	fuze_prox.c	fuze_prox.c		fuze_prox.c	fuze_prox.c	ruze_prox.c			ruze_prox.c	fuze_prox.c		ruze_prox.c			J XUJU OZIIJ	nac_prov.	fuze prox.c	វី	fuze_prox.c
2nd 3rd 4th 5th 6th 7th 8th store_traj_clord event_get_eventid vec_copy d2f_vec_copy		flechette veh_list	missile_flechette_speed_coeff	missile_util_eval_poiy  vec_scale	vec_aaa mil VehicleID	missile fuze all prox	missile fuze prox	missile_fuze_invest_prox	print near get next_vel_near_point	get prox	free_ptr	prox_free	malloc near get veh if still_near_point	free_prox	free_ptr	printl prox_free	free	missile_fuze_detonate_prox	vec_scale	printf	free_prox	free_ptr	printf	AND THE STATE OF T	f2d_vec_scale

	fuze_prox.c util_comm.c util_comm.c	util_comm.c	util_comm.c sun_stubs.c	sun_wayed.c	sub_flech.c util_comm.c	sun_stubs.c	util_comm.c	sun_wayed.c	fuze_prox.c	fuze_prox.c fuze_prox.c	fuze_prox.c
3rd 4th 5th 6th 7th 6th vec_sub vec_dot_vrod	vec_add f2d_mat_transpose missile_util_comm_fuze_detonate missile_comm	vec_mat_mul missile_util_comm_check_sub_mun	printf missile_comm veh_kinematics	kinenatics_range_squared network_ifire_init_burst network_ifire_send_detonation map_get_ammo_entry_from_network_type	npacts_larene_spect network_send_vehicle_impact zero_vector missile_util_comm_release_sub_munition	veh_kinematics	kinenutics_runge_squarea missile_comm store_traj_chord event_get_eventid	vec_copy d2f_vec_copy map_get_ammo_entry_from_network_type network_send_projectile_fire_pkt	impacts_queue_effect missile_fuze_detonate_prox vec_scale	printf free_prox free_ptr	printf prox_free free

RWA AIRNET CALL TREE STRUCTURE 6th 7th 8th

5th

4th

3rd

2nd

fuze_prox.c	fuze_prox.c util_comm.c util_comm.c	fuze_prox.c fuze_prox.c fuze_prox.c	fuze_prox.c rkt_hydra.c	rkt_hydra.c rkt_hydra.c rwa_hydra.c	rwa_hydra.c rwa_hydra.c	rwa_stubs.c	rwa_main.c	rwa_sound.c rwa_sound.c
RWA AIRNET CALL TREE STRUCTURE  1st 2nd 3rd 4th 5th 6th 7th 8th  2d_vec_scale  vec_sub	vec_dot_prod vec_add f2d_mat_transpose missile_util_comm_fuze_detonate missile_comm	vec_mat_mul missile_fuze_prox_stop free_prox free_ptr vrintf	prox_free free network_ifire_send_indirect_fire	missile_hydra_purge_iree_imssires rkts_in_flight hydra_fly pylons_set nvlon R	rotate_set_no_rotate  pylon_L articulation left_rocket_launch hydra_launch_rocket right_rocket_launch	Lrf Post Lrf Err String fprintf RTC_REPAIR_SIMUL repair_simul RTC_NET_SIMUL	io_simul velt spec stop	idc_reset sound_reset dont_use_sound

	rwa_mem.c	rwa_sound.c	rwa_sound.c	I Wa_VISIOII.C	rwa_vision.c		rwa_sound.c	rwa_mem.c	rwa_sound.c	rwa_sound.c	rwa_main.c		sun_stubs.c
RWA AIRNET CALL TREE STRUCTURE	1st 2nd 3rd 4th 5th 6th 7th 8th sounds	fifo_enqueue sound_crror	ffush sound_array	vision_break_all_blocks clear_view_flags get_cig2_present	get_cig2-present vision_clear_tc_board clear_view_flags	Lrf Un Init Lrf Err String fprintf	hull_uninit sound_reset	dont_use_sound sounds	fifo_enqueue sound_error	fprintf fftush veh_sound_array	cig_uninit dtad_uninit bbd_uninit	veh_spec_exit keyboard_exit_gracefully rwa_config_exit_gracefully vision_break_all_blocks timers_get_current_time	printf timers_get_current_tick timers_elapsed_milliseconds network_print_statistics net_handle net_close
	Ä												

RWA AIRNET CALL TREE STRUCTURE 6th 7th 8th

5th 2nd 3rd 4th 5th mem\_free\_shared\_memory reboot\_on\_shutdown

**1st** 

- A-45 -

# Appendix B - Source code listing for rwa\_aerodyn.c.

The following appendix contains the source code listing for rwa\_aerodyn.c for convenience in document maintenance and understanding of the CSU.

```
/* $Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/rwa/src/RCS/rwa_aerodyn.c,v 1.6
1993/01/28 23:33:00 cm-adst Exp $ */
 * $Log: rwa_aerodyn.c,v $
 * Revision 1.6 1993/01/28 23:33:00 cm-adst
 * P. DesMeueles's changes for spcr 31
 * Revision 1.5 1992/12/21 22:14:41 cm-adst
 * R. Branson's flight changes. These changes will become
 * BDS-D 1.1.1. This change was turned over by C. Swanson.
 * Revision 1.1 1992/10/07 19:00:23 cm-adst
 * Initial Version
static char RCS_ID[] = "$Header: /a3/adst-
cm/RWA/AIRNET/simnet/vehicle/rwa/src/RCS/rwa_aerodyn.c,v 1.6 1993/01/28 23:33:00
cm-adst Exp $";
   ********************
* Revisions:
                                                               SP/CR Number
                           Author
                                        Title
                   Date
      Version
                   10/09/92 R. Branson Data File Initiali-
      1.2
                                          zation
                  10/16/92 R. Branson Data filenames changed
      1.3
                                          to eight charachters
                   10/30/92 R. Branson Added pathname to data
      1.4
                                          directory
                   01/19/93 P.Desmeules Increased the size of the
      1.5
                                        fgets to make sure the
                                        whole line is read in.
                                                                     85
                   03/04/93 P.Desmeules Fix the value of
      1.5
                                        HOVER_AUG_PITCH_RESET_VALUE
      SP/CR No. Description of Modification
                  Hard coded defines changed to array elements.
                  Aerodyn data array added.
                  Aerodyn initialization data array added.
                  Aerodyn stealth data array added.
                  Aerodyn simple data array added.
                  Added file read for aerodyn data, aerodyn initiali-
                     zation data, aerodyn stealth data, and aerodyn
                     simple data to the "aerodyn_init" function.
                  Added "/simnet/data/" to each data file pathname.
```

```
/**********************
                rwa_aerodyn.c
* FILE:
               James Chung
* AUTHOR:
* MAINTAINER: James Chung
                4/19/89 james: Creation
 * HISTORY:
                 8/02/90 carol: added simplified aero dynamics*
* Copyright (c) 1989 BBN Systems and Technologies Corporation
 * All rights reserved.
* Interim aerodynamics model for a generic rotary-wing aircraft*
 * with flight characteristics similar to that of a McDonnell
 * Douglas AH-64 Apache attack helicopter.
 ****************
#include "stdio.h"
#include "simstdio.h"
#include "math.h"
#include "sim_dfns.h"
#include "sim_types.h"
#include "sim_macros.h"
#include "libmatrix.h"
#include "libmath.h"
#include "rwa_engine.h"
#include "vehicle.h"
#include "aero_param.h"
#include "std_atm.h"
#include "ground.h"
#include "rwa_ground.h"
#include "parameters.h"
#include "rwa_kinemat.h"
#include "libmun.h"
#include "libhull.h"
#include "libkin.h"
#include "rwa_aerodyn.h"
* Debug macro
#ifdef FILEDBG
#define P(a)
#else
#define P(a)
#endif
                                       aero_data[ 0]
#define MOMENT_OF_INERTIA_X
                                       aero_data[ 1]
#define MOMENT_OF_INERTIA_Y
                                        aero_data[ 2]
#define MOMENT_OF_INERTIA_Z
```

```
aero_data[ 3]
#define AIRFRAME_MASS
                                          aero_data[ 4]
#define ORDINANCE_MASS
                                          aero_data[ 5]
#define GRAV_CONSTANT
                                          aero data[ 6]
#define CG_AC_X
                                          aero_data[ 7]
#define CG_AC_Y
                                          aero_data[ 8]
#define CG_AC_Z
                                          aero_data[ 9]
#define VIRTUAL_WING_AREA
                                          aero_data[10]
#define VIRTUAL_WING_COP_AC_X
                                         aero_data[11]
#define VIRTUAL_WING_COP_AC_Y **
                                         aero_data[12]
#define VIRTUAL_WING_COP_AC_Z
                                        aero_data[13]
aero data[14]
#define WING_LIFT_COEFFICIENT_FIT_3
                                         aero_data[14]
#define WING_LIFT_COEFFICIENT_FIT_2
                                     aero_data[15]
aero_data[16]
#define WING_LIFT_COEFFICIENT_FIT_1
#define WING_LIFT_COEFFICIENT_FIT_0
                                 (deg_to_rad(aero_data[17]))
#define WING_STALL_AOA
                                          aero_data[18]
#define VSTAB_AREA
                                          aero_data[19]
#define VSTAB_COP_AC_X
                                          aero_data[20]
#define VSTAB_COP_AC_Y
                                          aero_data[21]
#define VSTAB_COP_AC_Z
                                          aero_data[22]
#define VSTAB_LIFT_COEFFICIENT_1
                                  (deg_to_rad(aero_data[23]))
#define VSTAB_STALL_SSA
                                          aero_data[24]
#define MAIN_ROTOR_COP_AC_X
                                          aero_data[25]
#define MAIN_ROTOR_COP_AC_Y
                                          aero_data[26]
#define MAIN_ROTOR_COP_AC_Z
#define MAIN_ROTOR_MAX_THRUST
                                          aero_data[27]
                                  (deg_to_rad(aero_data[28]))
#define MAIN_ROTOR_MAST_TILT
                                        aero_data[29]
#define MAIN_ROTOR_MAX_LOAD_TORQUE
                                          aero_data[30]
#define MAIN_ROTOR_MAX_PITCH_MOMENT
                                     aero_data[31]
#define MAIN_ROTOR_MAX_ROLL_MOMENT
#define MAIN_ROTOR_TORQUE_COUPLING_GAIN aero_data[32]
#define MAIN_ROTOR_GROUND_EFFECT_FACTOR aero_data[33]
                                          aero_data[34]
#define TAIL_ROTOR_COP_AC_X
                                          aero_data[35]
#define TAIL_ROTOR_COP_AC_Y
                                          aero_data[36]
#define TAIL_ROTOR_COP_AC_Z
                                          aero_data[37]
#define TAIL_ROTOR_MAX_THRUST
                                         aero_data[38]
#define TAIL_ROTOR_MAX_LOAD_TORQUE
                                          aero_data[39]
#define P_DRAG_COEFF_CONST
                                        aero_data[40]
#define P_DRAG_TAS_BREAK
                                          aero_data[41]
#define P_DRAG_COEFF_BREAK
                                          aero_data[42]
#define P_DRAG_TAS_MAX
                                          aero_data[43]
#define P_DRAG_COEFF_MAX
                                         aero_data[44]
#define TOTAL_WETTED_SURFACE_AREA
#define ATT_DAMPING_MODE_SIMPLE TRUE
 Hover hold changes:
if ATT DAMPING_MODE_SIMPLE
      when slow moving ( airspeed<10 knots ) the max pitch is 5 degrees
```

```
( 10<=airspeed<30 ) pitch is 10 degrees
          medium
                                          pitch is 15 degrees
                     ( 30<=airspeed )
          other
else
     when airspeed >= 10 knots pitch is proportional to log(speed)
     otherwise pitch is +/- 5 degrees
                           Paul J. Metzger 11-1-89
static REAL MAX_ATT_CTL_ANGLE;
                                    aero_data[45]
aero_data[46]
aero_data[47]
aero_data[48]
/* transition mode, TRUE or FALSE */
#define MAX_ATT_CTL_ANGLE_STOP
#define MAX_ATT_DAMPING_FACTOR
#define HOVER_SLOW_LIMIT
#define HOVER_AUG_PITCH_RESET_VALUE
static int hover_hold_turned_on;
#if ATT_DAMPING_MODE_SIMPLE
                                    (deg_to_rad (aero_data[49]))
#define MAX_ATT_CTL_ANGLE_NORM
                                (deg_to_rad (aero_data[50]))
#define MAX_ATT_CTL_ANGLE_MED
                                  (deg_to_rad (aero_data[51]))
#define MAX_ATT_CTL_ANGLE_SLOW
                                        aero_data[52]
#define HOVER_MED_LIMIT
#endif
                                        aero_data[53]
#define ATT_CTL_PITCH_P_GAIN
                                       aero_data[54]
#define ATT_CTL_PITCH_I_GAIN
                                       aero_data[55]
#define ATT_CTL_ROLL_P_GAIN
                                        aero_data[56]
#define ATT_CTL_ROLL_I_GAIN
                                      aero_data[57]
#define HOVER_AUG_ROLL_P_GAIN
                                       aero_data[58]
#define HOVER_AUG_ROLL_I_GAIN
                                      aero_data[59]
aero_data[60]
#define HOVER_AUG_PITCH_P_GAIN
#define HOVER_AUG_PITCH_I_GAIN
                                       aero_data[61]
#define HOVER_AUG_YAW_P_GAIN
                                        aero_data[62]
                                  aero_data[63]
#define HOVER_AUG_YAW_I_GAIN
#define HOVER_AUG_CLIMB_P_GAIN
                                        aero_data[64]
#define HOVER_AUG_CLIMB_I_GAIN
#define MAX_STAB_AUG_PITCH_ROLL_CONTROL aero_data[65]
                                        aero_data[66]
#define MAX_STAB_AUG_YAW_CLIMB_CONTROL
                                         aero_data[67]
#define ROLL_RATE_DAMPING_GAIN
                                        aero data[68]
#define PITCH_RATE_DAMPING_GAIN
                                        aero_data[69]
#define YAW_RATE_DAMPING_GAIN
                                        aero_data[70]
#define VERTICAL_RATE_DAMPING_GAIN
#define LATERAL_VELOCITY_DAMPING_GAIN
                                         aero_data[71]
#define LIFT_COEFF_VIRTUAL_WING
                                        aero_data[72]
                                        aero_data[73]
#define OSWALD_EFFIC_FACTOR
                                         aero_data[74]
#define INDUCED_DRAG_COEFF
* SPCR 85 - fix the value of HOVER_AUG_PITCH_RESET (element 48) from
 * .44 to .044
 static REAL aero_data[100] = {
      50000.000, 50000.000, 50000.000, 4881.000, 1591.000,
```

```
25.0,
                0.0, 0.0,
0.0, 0.0,
                                     -0.100,
        9.8,
                                     0.0,
3.0,
                                                 0.0,
                0.0,
        0.0,
                                                 0.0,
                0.0,
                          30.0,
        1.0,
                                     60.0,
                                                 0.0,
                          5.0,
                0.0,
       -9.1,
                                   2.5,
                                              76476.0,
                2.0, 123500.0,
        0.0,
                                              0.0,
      100000.0, 100000.0, 0.5,
       00000.0, 100000.0, 0.5, 0.4,

-9.1, 0.0, 8909.1, 1684.8,

50.0, 0.02, 100.0, 0.06,
                                                 0.0,
                                                 50.0.
                                    0.044,
2.5,
0.001,
                                                 15.0,
                4.5, 5.15,
        6.0,
                         15.46,
                                                 0.05.
       10.0,
                 6.0,
                         0.1,
                                   1.0,
                                                 0.1.
                 0.05,
        5.0,
                 10.0, 5.0, 1.0, 0.05, 100000.0, 100000.0,
                                                 0.5,
                10.0,
        0.001,
                                             100000.0,
        0.2,
                                 0.9,
                                               0.0,
               1000.0, 0.6,
      2000.0,
                                      0.0,
                                                 0.0,
                           0.0,
                0.0,
        0.0,
                                      0.0,
                                                 0.0,
                           0.0,
        0.0,
                  0.0,
                   0.0, 0.0,
0.0, 0.0,
                                                  0.0.
                                      0.0,
                 0.0,
         0.0,
                                        0.0,
                                                  0.0,
          0.0,
                                                  0.0
                                       0.0,
                  0.0,
        0.0,
       } ;
static REAL aero_init[20] = {
                                                 0.0,
                            0.0,
                                       0.0,
        0.0, 0.0,
                                                 0.0,
                  0.0,
                           0.0,
                                      0.0,
         0.0,
                             0.0,
                                        0.0,
                                                   0.0,
                   0.0,
          0.0,
                                                 0.0
                                       0.0,
                            0.0,
                  0.0,
        0.0.
       } ;
static REAL aero_simple[20] = {
      500000.0, 0.5, 48.0,
                                       0.15,
                                                 10.0,
                                                  0.03.
                                        0.7,
                            1.5,
0.0,
         100.0, 150000.0,
                                                   0.0,
                                        0.0,
      400000.0, 100.0,
                                                 0.0
                            0.0,
                                      0.0,
                   0.0,
         0.0,
static REAL aero_stealth[20] = {
        80.0, 30.0, 10.0, 1000000000.0, 1000000000.0,
                          0.03, 0.0, 0.0,
                 25000.0,
        5000.0,
                              0.0,
                                                   0.0,
                                         0.0,
          0.0, 0.0,
                                                  0.0
                                       0.0,
                           0.0,
                   0.0,
         0.0,
       } ;
                                    /* OFF or ON */
static int hover_hold_state;
static REAL MAIN_ROTOR_MAST_TILT_SIN;
static REAL MAIN_ROTOR_MAST_TILT_COS;
                                     /* m */
static REAL altitude;
                                     /* m/sec */
static REAL true_airspeed;
                                   /* m/sec */
static REAL last_airspeed = 0;
                                     /* m/sec */
static REAL vertical_speed;
                                     /* rad */
static REAL roll;
                                     /* rad */
static REAL pitch;
                                     /* rad/sec */
static REAL roll_rate;
                                     /* rad/sec */
static REAL pitch_rate;
```

```
static REAL g_force;
static REAL last_g_force;
                                          /* rad/sec */
static REAL yaw_rate;
static REAL pitch_damping;
static REAL roll_damping;
static REAL yaw_damping;
                                        /* deg R */
static REAL ambient_temperature;
                                          /* N / m^2 */
static REAL ambient_pressure;
                                         /* kg / m^3 */
static REAL ambient_density;
                                         /* N / m^2 */
static REAL dynamic_pressure;
                                                /* N */
static REAL main_rotor_thrust;
                                                 /* N */
static REAL tail_rotor_thrust;
                                                /* N */
static REAL lift_virtual_wing;
static REAL lift_vstab;
static REAL lift_coefficient_virtual_wing;
static REAL lift_coefficient_vstab;
static REAL total_drag;
static REAL total_incompressible_drag_coefficient;
                                         /* N */
 static REAL gross_weight;
                                          /* kg */
 static REAL vehicle_mass;
                                          /* rad */
 static REAL angle_of_attack;
                                          /* rad */
 static REAL side_slip_angle;
                                          /* N-m */
 static REAL main_rotor_load_torque;
                                        /* N-m */
 static REAL tail_rotor_load_torque;
 static REAL powertrain_percent_shaft_speed;
                                                      /* Flight controls */
                                    /* -1 to 1 */
 static REAL cyclic_pitch;
                                    /* -1 to 1 */
/* 0 to 1 */
                                                       /* Flight controls */
 static REAL cyclic_roll;
 static REAL collective;
                                    /* -1 to 1 */
 static REAL pedal;
 static REAL stab_aug_pitch;
 static REAL stab_aug_roll;
 static REAL stab_aug_yaw;
 static REAL stab_aug_climb;
 static REAL stab_aug_pitch_integrator;
 static REAL stab_aug_roll_integrator;
 static REAL stab_aug_yaw_integrator;
 static REAL stab_aug_climb_integrator;
 static REAL hover_aug_pitch_angle;
. static REAL hover_aug_roll_angle;
 static REAL hover_aug_pitch_integrator;
 static REAL hover_aug_roll_integrator;
 static REAL attitude_control_roll_integrator;
 static REAL attitude_control_pitch_integrator;
 static REAL attitude_control_roll_command;
 static REAL attitude_control_pitch_command;
 static REAL controller_cyclic_pitch;
 static REAL controller_cyclic_roll;
 static REAL controller_collective;
 static REAL controller_tail_rotor;
                                            /* kinematic state vectors */
 static REAL *angular_velocity_vector;
 static REAL *normalized_velocity_vector;
 static REAL *velocity_vector;
  static REAL *gravity_dir_vector;
```

```
static REAL p_drag_fit_coeff[9]; /* parasite drag fit coefficients */
static REAL oswald_efficiency_factor;
static REAL induced_drag_coefficient;
static REAL parasite_drag_coefficient;
static VECTOR loc_ac_main_rotor_cop;
static VECTOR loc_ac_tail_rotor_cop;
static VECTOR loc_ac_virtual_wing_cop;
static VECTOR loc_ac_vstab_cop;
static VECTOR loc_ac_cg;
                                              /* body [X Y Z] */
static VECTOR lift_body_virtual_wing;
static VECTOR lift_body_vstab;
static VECTOR force_body_main_rotor;
static VECTOR force_body_tail_rotor;
static VECTOR force_body_damping;
static VECTOR drag_body;
static VECTOR gravity_force_body;
static VECTOR force_ground_effect;
                                         /* sum of all forces */
static VECTOR force_body;
                                              /* body [X Y Z] */
static VECTOR moment_body_virtual_wing;
static VECTOR moment_body_vstab;
static VECTOR moment_body_main_rotor;
static VECTOR moment_body_torque_coupling;
static VECTOR moment_body_tail_rotor;
static VECTOR moment_body_cg;
static VECTOR moment_body_damping;
static VECTOR moment_body;
static VECTOR virtual_wing_force; /* velocity [H D L] */
static VECTOR vstab_force;
static VECTOR drag_force;
                                    /* vel -> body xform */
static T_MAT_PTR velocity_to_body;
static T_MATRIX inertia_matrix =
{ {50000.0, 0, 0},
  {0, 50000.0, 0},
  {0, 0, 50000.0}};
int funny_little_kludge = 1;/* default is logarithmic for complex model */
static int aerodyn_debug = 0;
static int selected_model = COMPLEX_MODEL; /* default: James' model */
static int allow_takeoff = TRUE; /* allow stealth model to take off */
                                   /* unset any pitch */
static int level_view = TRUE;
static REAL ground_height = 2.8;
void aero_body_point_set_front_wheels( distance_from_hull )
REAL distance_from_hull;
    body_point[0].position[Z] = distance_from_hull;
```

```
body_point[1].position[Z] = distance_from_hull;
   ground_height = (REAL)(((int)(-distance_from_hull * 10)) / 10.0);
   printf( "Front Wheels set %1.41f m. under Hull.\n",
           distance_from_hull);
}
void aero_body_point_set_rear_wheel( distance_from_hull )
REAL distance_from_hull;
   body_point[2].position[Z] = distance_from_hull;
    printf( "Rear Wheel set %1.41f m. under Hull.\n",
           distance_from_hull);
}
REAL aero_get_ground_height()
    return( ground_height );
void aerodyn_init()
   int i;
                                                                   */
/* DEFAULT DATA FOR rwa_aerodyn.c READ FROM FILE
    int
            j;
    float
            data_tmp;
            descript[80];
    char
    FILE
            *fp;
      P(printf("$$$$ RWA AERODYN $$$$\n"););
        fp = fopen("/simnet/data/rwa_aero.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/rwa_aero.d\n");
            exit();
      rewind(fp);
            Read array data
      j=0;
      while(fscanf(fp, "%f", &data_tmp) != EOF){
            aero_data[j] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("aero_data(%3d) is%11.3f %s\n", j, aero_data[j],
                   descript););
            ++j;
            }
      fclose(fp);
                                                                    */
/* END DEFAULT DATA FOR rwa_aerodyn.c READ FROM FILE
/* DEFAULT INITIALIZATION DATA FOR rwa_aerodyn.c READ FROM FILE */
```

```
fp = fopen("/simnet/data/rw_ae_in.d", "r");
     if(fp==NULL){
           fprintf(stderr, "Cannot open /simnet/data/rw_ae_in.d\n");
           exit();
           }
     rewind(fp);
           Read array data
                            */
     i=0;
     while(fscanf(fp, "%f", &data_tmp) != EOF){
           aero_init[j] = data_tmp;
           fgets(descript, 80, fp);
           P(printf("aero_init(%3d) is%11.3f %s\n", j, aero_init[j],
                 descript););
           ++j;
           }
     fclose(fp);
   END DEFAULT INITIALIZATION DATA FOR rwa_aerodyn.c READ FROM FILE
  DEFAULT SIMPLE INITIALIZATION DATA FOR rwa_aerodyn.c READ FROM FILE */
     fp = fopen("/simnet/data/rw_ae_sp.d", "r");
     if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/rw_ae_sp.d\n");
            exit();
            }
     rewind(fp);
           Read array data
     j=0;
     while(fscanf(fp, "%f", &data_tmp) != EOF){
            aero_simple[j] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("aero_simple(%3d) is%11.3f %s\n", j, aero_simple[j],
                  descript););
            ++j;
      fclose(fp);
  END DEFAULT SIMPLE INITIALIZATION DATA FOR rwa_aerodyn.c READ FROM FILE*/
/* DEFAULT STEALTH INITIALIZATION DATA FOR rwa_aerodyn.c READ FROM FILE
      fp = fopen("/simnet/data/rw_ae_sl.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/rw_ae_sl.d\n");
            exit();
      }
     rewind(fp);
            Read array data
                               */
```

```
j=0;
     while(fscanf(fp, "%f", &data_tmp) != EOF){
           aero_stealth[j] = data_tmp;
           fgets(descript, 80, fp);
           P(printf("aero_stealth(%3d) is%11.3f %s\n", j, aero_stealth[j],
                 descript););
           ++j;
     }
     fclose(fp);
/* END DEFAULT STEALTH INITIALIZATION DATA FOR rwa_aerodyn.c READ FROM FILE*/
   engine_init();
   cyclic_pitch =
                                    aero_init[ 0];
   cyclic_roll =
                              aero_init[ 1];
   if (selected_model != STEALTH_MODEL)
                                    aero_init[ 2];
        collective =
   else
       collective = 0.5;
     allow_takeoff = TRUE;
    }
                                    aero_init[ 3];
   pedal =
                                    aero_init[ 4];
   stab_aug_pitch_integrator =
                                          aero_init[5];
   stab_aug_roll_integrator =
                                          aero_init[ 6];
   stab_aug_yaw_integrator =
                                    aero_init[ 7];
   stab_aug_climb_integrator =
   attitude_control_pitch_integrator = aero_init[ 8];
   attitude_control_roll_integrator =
                                          aero_init[ 9];
   hover_aug_pitch_integrator =
                                    aero_init[10];
   hover_aug_roll_integrator =
                                    aero_init[11];
                                    aero_init[12];
   hover_aug_pitch_angle =
   hover_aug_roll_angle =
                                    aero_init[13];
   hover_hold_state = OFF;
   hover_hold_turned_on = FALSE;
    loc_ac_main_rotor_cop[X] = MAIN_ROTOR_COP_AC_X;
    loc_ac_main_rotor_cop[Y] = MAIN_ROTOR_COP_AC_Y;
    loc_ac_main_rotor_cop[Z] = MAIN_ROTOR_COP_AC_Z;
    loc_ac_tail_rotor_cop[X] = TAIL_ROTOR_COP_AC_X;
    loc_ac_tail_rotor_cop[Y] = TAIL_ROTOR_COP_AC_Y;
    loc_ac_tail_rotor_cop[Z] = TAIL_ROTOR_COP_AC_Z;
    loc_ac_virtual_wing_cop[X] = VIRTUAL_WING_COP_AC_X;
    loc_ac_virtual_wing_cop[Y] = VIRTUAL_WING_COP_AC_Y;
    loc_ac_virtual_wing_cop[Z] = VIRTUAL_WING_COP_AC_Z;
    loc_ac_vstab_cop[X] = VSTAB_COP_AC_X;
    loc ac vstab cop[Y] = VSTAB_COP_AC_Y;
    loc_ac_vstab_cop[Z] = VSTAB_COP_AC_Z;
```

```
loc_ac_cg[X] = CG_AC_X;
   loc_ac_cg[Y] = CG_AC_Y;
   loc_ac_cg[Z] = CG_AC_Z;
   inertia_matrix[1] [1] = MOMENT_OF_INERTIA_X;
   inertia_matrix[2] [2] = MOMENT_OF_INERTIA_Y;
   inertia_matrix[3] [3] = MOMENT_OF_INERTIA_Z;
   pitch_damping = PITCH_RATE_DAMPING_GAIN;
   roll_damping = ROLL_RATE_DAMPING_GAIN;
   yaw_damping = YAW_RATE_DAMPING_GAIN;
   MAIN_ROTOR_MAST_TILT_SIN = sin(MAIN_ROTOR_MAST_TILT);
   MAIN_ROTOR_MAST_TILT_COS = cos(MAIN_ROTOR_MAST_TILT);
   vec_init (vstab_force);
   vec_init (drag_force);
   vec_init (ground_force);
   vec_init (force_ground_effect);
   vec_init (force_body);
   vec_init (moment_body);
  vec_init (moment_body_torque_coupling);
   vec_init (force_body_main_rotor);
   vec_init (force_body_tail_rotor);
   vec_init (force_body_damping);
   vehicle_mass_init (AIRFRAME_MASS + ORDINANCE_MASS, inertia_matrix );
   ground_init();
                                   /* Set parasite drag profile */
   for (i=0; i<9; i++)
     p_drag_fit_coeff[i] = 0.0;
    if (find_cubic_func (0.0, P_DRAG_COEFF_CONST,
                  P_DRAG_TAS_BREAK, P_DRAG_COEFF_BREAK,
                  P_DRAG_TAS_MAX, P_DRAG_COEFF_MAX,
                  0.5, p_drag_fit_coeff) != TRUE)
      fprintf (stderr, "AERODYN: Error - unable to fit p_drag function\n");
/* So one can tweak the constants without recompiling */
    if (selected_model)
        aerodyn_read_simple_constants (get_constants_file ());
}
static void get_aircraft_kinematic_state()
    orientation_calc();
    parameters_calc();
```

```
true_airspeed = kinematics_get_true_airspeed();
   altitude = kinematics_get_altitude();
   angular_velocity_vector = kinematics_get_angular_velocity_vector();
   normalized_velocity_vector = kinematics_get_normalized_velocity_vector();
   velocity_vector = kinematics_get_linear_velocity_vector();
   gravity_dir_vector = kinematics_get_gravity_vector();
   angle_of_attack = kinematics_get_aoa();
   side_slip_angle = - kinematics_get_yaw();
   velocity_to_body = kinematics_get_velocity_to_body();
   g_force = kinematics_get_g_force();
   vertical_speed = kinematics_get_vertical_speed();
}
static void deb_mat_print (m)
    T_MATRIX m;
    int i;
    for (i=0; i<=2; i++)
        printf("%0.3lf %0.3lf %0.3lf\n", m[i][0], m[i][1], m[i][2]);
}*
static void compute_flight_parameters()
    ambient_density = air_density(altitude);
    ambient_temperature = air_temperature(altitude);
    ambient_pressure = air_pressure(altitude);
    dynamic_pressure = 0.5 * ambient_density * square (true_airspeed);
    pitch_rate = angular_velocity_vector[X];
    roll_rate = angular_velocity_vector[Y];
    yaw_rate = angular_velocity_vector[2];
    roll = atan2 (-gravity_dir_vector[X], -gravity_dir_vector[Z]);
    pitch = atan2 (-gravity_dir_vector[Y], -gravity_dir_vector[Z]);
}
static void interact_with_ground()
    REAL brake_factor;
    brake_factor = normalized_velocity_vector[Y] *
                  true_airspeed / (true_airspeed + 5);
    body_point[0].x_force = - 6000 * brake_factor;
    body_point[1].x_force = body_point[0].x_force;
    ground_interaction(ground_force,ground_torque,body_point,grnd,
            NUMBER_OF_BODY_POINTS);
     force_ground_effect[Z] = main_rotor_thrust
                   * MAIN_ROTOR_GROUND_EFFECT_FACTOR
                   / (cig_altitude_above_gnd() + 1.0);
 }
 /*******************
 /* fuel get current level returns gallons
```

```
/* gals * (6.5 lbs / gal) * (1kg / 2.2 lbs)
#define KILOGRAMS_PER_GALLON 2.95454545454
static void compute_gross_weight()
    vehicle_mass = AIRFRAME_MASS + ORDINANCE_MASS +
      fuel_get_current_level() * KILOGRAMS_PER_GALLON;/* kg */
    gross_weight = vehicle_mass * GRAV_CONSTANT;
                                                      /* N */
}
void aerodyn_set_lateral_stick (val)
    REAL val;
{
    cyclic_roll = -val;
}
void aerodyn_set_longitudinal_stick (val)
    REAL val;
 cyclic_pitch = -val;
void aerodyn_set_pedal (val)
    REAL val;
{
    pedal = val;
}
void aerodyn_set_collective (val)
    REAL val;
{
    if (funny_little_kludge)
      collective = log10 (val * 9.0 + 1.0); /* or, how to make linear log */
    else
        collective = val;
}
static void compute_lift_drag_forces()
    lift_virtual_wing = dynamic_pressure *
            lift_coefficient_virtual_wing * VIRTUAL_WING_AREA;
    lift_vstab = dynamic_pressure * lift_coefficient_vstab * VSTAB_AREA;
    total_drag = total_incompressible_drag_coefficient * dynamic_pressure *
                  TOTAL_WETTED_SURFACE_AREA;
}
static void compute_body_damping_forces_and_moments()
    moment_body_damping[X] = - pitch_damping * pitch_rate;
    moment_body_damping[Y] = - roll_damping * roll_rate;
    moment_body_damping[Z] = - yaw_damping * yaw_rate;
```

```
force_body_damping[X] = -velocity_vector[X] * LATERAL_VELOCITY_DAMPING_GAIN;
    force_body_damping[Y] = 0.0;
    force_body_damping[Z] = -velocity_vector[Z] * VERTICAL_RATE_DAMPING_GAIN;
}
static REAL virtual_wing_lift_coefficient (alpha)
    REAL alpha;
    if (alpha > WING_STALL_AOA || alpha < 0.0)
      return (0.0);
    else
             (((WING_LIFT_COEFFICIENT_FIT_3 * alpha +
    return
              WING_LIFT_COEFFICIENT_FIT_2) * alpha +
              WING_LIFT_COEFFICIENT_FIT_1) * alpha +
              WING_LIFT_COEFFICIENT_FIT_0);
}
static REAL vstab_lift_coefficient (yaw)
    REAL yaw;
{
    REAL yawval;
    if (abs(yaw) > VSTAB_STALL_SSA)
      yawval = sign(yawval) * VSTAB_STALL_SSA;
    else
      yawval = yaw;
    return (VSTAB_LIFT_COEFFICIENT_1 * yawval);
}
static void compute_lift_drag_coefficients()
    REAL multiplier;
    lift_coefficient_vstab = vstab_lift_coefficient (side_slip_angle);
/* Computing virtual wing coefficient as independent of AOA */
    lift_coefficient_virtual_wing = LIFT_COEFF_VIRTUAL_WING;
                  virtual_wing_lift_coefficient (angle_of_attack); */
    parasite_drag_coefficient = cubic_func (true_airspeed, p_drag_fit_coeff);
    if (true_airspeed > 0.0 && angle_of_attack > 0.0) /* speed brake */
      multiplier = 5.0 * true_airspeed * sin(angle_of_attack);
       if (multiplier > 1.0)
           parasite_drag_coefficient *= multiplier;
     }
     oswald_efficiency_factor = OSWALD_EFFIC_FACTOR;
     induced_drag_coefficient = INDUCED_DRAG_COEFF;
     total_incompressible_drag_coefficient = parasite_drag_coefficient +
                                induced_drag_coefficient;
```

```
}
static void send_to_dynamics_kinematics()
    vehicle_mass_init (vehicle_mass, inertia_matrix);
    vehicle_forces (force_body);
    vehicle_torques (moment_body);
}
void dump_forces()
    vec_dump ("lift_body_virtual_wing:", lift_body_virtual_wing);
    vec_dump ("lift_body_vstab:", lift_body_vstab);
    vec_dump ("drag_body:", drag_body);
    vec_dump ("gravity_force_body:", gravity_force_body);
    vec_dump ("force_body_main_rotor:", force_body_main_rotor);
    vec_dump ("force_body_tail_rotor:", force_body_tail_rotor);
    vec_dump ("ground_force:", ground_force);
    vec_dump ("force_body:", force_body);
static void sum_body_forces_and_moments_about_ac()
    vec_init (force_body);
    vec_add (force_body, force_body_main_rotor, force_body);
      vec_add (force_body, force_body_tail_rotor, force_body); */
    vec_add (force_body, lift_body_virtual_wing, force_body);
    vec_add (force_body, lift_body_vstab, force_body);
    vec_add (force_body, drag_body, force_body);
    vec_add (force_body, force_body_damping, force_body);
    vec_add (force_body, gravity_force_body, force_body);
    vec_add (force_body, ground_force,force_body);
    vec_add (force_body, force_ground_effect, force_body);
    vec_cross_prod(loc_ac_tail_rotor_cop, force_body_tail_rotor,
                         moment_body_tail_rotor);
    vec_cross_prod(loc_ac_virtual_wing_cop,lift_body_virtual_wing,
                         moment_body_virtual_wing);
    vec_cross_prod(loc_ac_vstab_cop, lift_body_vstab, moment_body_vstab);
    vec_cross_prod(loc_ac_cg, gravity_force_body, moment_body_cg);
     vec_init (moment_body);
     vec_add (moment_body, moment_body_main_rotor, moment_body);
     vec_add (moment_body, moment_body_tail_rotor, moment_body);
     vec_add (moment_body, moment_body_virtual_wing, moment_body);
     vec_add (moment_body, moment_body_vstab, moment_body);
     vec_add (moment_body, moment_body_cg, moment_body);
     vec_add (moment_body, ground_torque, moment_body);
     vec_add (moment_body, moment_body_damping, moment_body);
 }
 static void transform_lift_drag_forces_to_body_coordinates()
     virtual_wing_force[Z] = lift_virtual_wing; /* [H, D, L] */
```

```
vstab_force[X] = lift_vstab;
        drag_force[Y] = -total_drag;
                                                                                                  /* jwc 8/90 */
        if (true_airspeed < P_DRAG_TAS_BREAK)</pre>
            drag_force[Y] -= sin(pitch) * 50000;
        vec_mat_mul (virtual_wing_force, velocity_to_body, lift_body_virtual_wing);
        vec_mat_mul (vstab_force, velocity_to_body, lift_body_vstab);
        vec_mat_mul (drag_force, velocity_to_body, drag_body);
}
static void generate_gravity_body_force()
        compute_gross_weight();
        gravity_force_body[X] = gravity_dir_vector[X] * gross_weight;
        gravity_force_body[Y] = gravity_dir_vector[Y] * gross_weight;
        gravity_force_body[Z] = gravity_dir_vector[Z] * gross_weight;
}
static int frame;
void aerodyn_debug_print()
        REAL roll, pitch, yaw, heading, airspeed_knots, weight_lbs, thrust_lbs;
         REAL *position;
        roll=atan2(-gravity_dir_vector[X],-gravity_dir_vector[Z]) *180.0 / 3.1416;
        pitch=atan2(-gravity_dir_vector[Y],-gravity_dir_vector[Z])*180.0 / 3.1416;
        yaw = side_slip_angle;
         airspeed_knots = true_airspeed * 3.26 / 1.69;
         weight_lbs = gross_weight / 9.8 * 2.2;
         position = vehicle_A_p();
         heading = rad_to_deg (kinematics_get_heading());
        printf ("KTAS = \$0.21f VV = \$0.31f \$0.31f \$0.31f YR = \$0.31f \n",
             airspeed_knots, velocity_vector[X], velocity_vector[Y],
             velocity_vector[Z], angular_velocity_vector[Z]);
        printf ("xyzh = %0.31f %0.31f %0.31f %0.21f rpy = <math>%0.31f %0.31f %0.31
             position[X], position[Y], position[Z], heading,
             roll, pitch, yaw);
         if (hover_hold_state == ON)
                 printf ("stab_aug[rpyc]: %0.31f %0.31f %0.31f %0.31f\n",
             stab_aug_roll, stab_aug_pitch, stab_aug_yaw, stab_aug_climb);
 }
 static void compute_rotor_loads()
         main_rotor_load_torque = controller_collective *
                                                      MAIN_ROTOR_MAX_LOAD_TORQUE;
         tail_rotor_load_torque = abs (controller_tail_rotor) *
                                                      TAIL_ROTOR_MAX_LOAD_TORQUE;
 }
 static void compute_engine_torque()
          engine_simul(main_rotor_load_torque, tail_rotor_load_torque, altitude);
```

```
powertrain_percent_shaft_speed = engine_get_rotor_percent_shaft_speed();
}
static void compute_rotor_forces_and_moments()
   main_rotor_thrust = powertrain_percent_shaft_speed * controller_collective
            * MAIN_ROTOR_MAX_THRUST;
    tail_rotor_thrust = powertrain_percent_shaft_speed * controller_tail_rotor
            * TAIL_ROTOR_MAX_THRUST;
    force_body_main_rotor[Y] = main_rotor_thrust * MAIN_ROTOR_MAST_TILT_SIN;
    force_body_main_rotor[Z] = main_rotor_thrust * MAIN_ROTOR_MAST_TILT_COS;
    force_body_tail_rotor[X] = tail_rotor_thrust;
    moment_body_main_rotor(X) =
            - controller_cyclic_pitch * MAIN_ROTOR_MAX_PITCH_MOMENT;
    moment_body_main_rotor[Y] =
            controller_cyclic_roll * MAIN_ROTOR_MAX_ROLL_MOMENT;
    moment_body_main_rotor[Z] =
            - main_rotor_load_torque * MAIN_ROTOR_TORQUE_COUPLING_GAIN;
static REAL limiter (lower, val, upper)
REAL lower, val, upper;
{
     if (val > upper) return (upper);
    else if (val < lower) return (lower);
    else return (val);
 }
static REAL set_roll_attitude (angle)
REAL angle;
    attitude_control_roll_integrator += ATT_CTL_ROLL_I_GAIN * (roll - angle);
     /**** These used to be attitude_control_pitch_integrator instead of
                                                  PJM
                                                         11-1-89
         attitude_control_roll_integrator.
     attitude_control_pitch_integrator =
             limiter (-0.1, attitude_control_pitch_integrator, 0.1);
     attitude_control_roll_integrator =
             limiter (-0.1, attitude_control_roll_integrator, 0.1);
     attitude_control_roll_command = ATT_CTL_ROLL_P_GAIN * (roll - angle);
     attitude_control_roll_command += attitude_control_roll_integrator;
                                      limiter (-MAX_STAB_AUG_PITCH_ROLL_CONTROL,
     attitude_control_roll_command =
                               attitude_control_roll_command,
                               MAX_STAB_AUG_PITCH_ROLL_CONTROL);
     return (attitude_control_roll_command);
 }
 static REAL set_pitch_attitude (angle)
 REAL angle;
     attitude_control_pitch_integrator +=
```

```
ATT_CTL_PITCH_I_GAIN * (pitch - angle);
   attitude_control_pitch_integrator =
           limiter (-0.1, attitude_control_pitch_integrator, 0.1);
   attitude_control_pitch_command = ATT_CTL_PITCH_P_GAIN * (pitch - angle);
   attitude_control_pitch_command += attitude_control_pitch_integrator;
   attitude_control_pitch_command = limiter (-MAX_STAB_AUG_PITCH_ROLL_CONTROL,
                              attitude_control_pitch_command,
                              MAX_STAB_AUG_PITCH_ROLL_CONTROL);
   return (attitude_control_pitch_command);
}
static void compute_stab_augmentation_gains()
    if (hover_hold_state == ON)
      if ( !hover_hold_turned_on )
            hover_hold_turned_on = TRUE ;
            pitch_damping = 2 * PITCH_RATE_DAMPING_GAIN; /* jwc 8/90 */
            roll_damping = 2 * ROLL_RATE_DAMPING_GAIN;
            /* You should already be "hovering" (airspeed < 10 knots)</pre>
               for hover hold to show little visible swaying. */
            hover_aug_roll_integrator = 0.0;
            hover_aug_pitch_integrator = HOVER_AUG_PITCH_RESET_VALUE ;
            stab_aug_yaw_integrator = 0.0;
            stab_aug_climb_integrator = 0.0;
#if ATT_DAMPING_MODE_SIMPLE
               (true_airspeed < HOVER_SLOW_LIMIT)
            if
                 if (true_airspeed > -HOVER_SLOW_LIMIT)
                  MAX_ATT_CTL_ANGLE = MAX_ATT_CTL_ANGLE_SLOW ;
                 else if (true_airspeed > -HOVER_MED_LIMIT)
                  MAX_ATT_CTL_ANGLE = MAX_ATT_CTL_ANGLE_MED;
                 else
                  MAX_ATT_CTL_ANGLE = MAX_ATT_CTL_ANGLE_NORM ;
             else if (true_airspeed < HOVER_MED_LIMIT)
                   MAX_ATT_CTL_ANGLE = MAX_ATT_CTL_ANGLE_MED ;
             else
                   MAX_ATT_CTL_ANGLE = MAX_ATT_CTL_ANGLE_NORM ;
 #endif
             }
 #if ATT_DAMPING_MODE_SIMPLE
       if (true_airspeed > HOVER_SLOW_LIMIT )
             MAX_ATT_CTL_ANGLE =
                   log( true_airspeed ) * MAX_ATT_DAMPING_FACTOR ;
       else if (true_airspeed < -HOVER_SLOW_LIMIT )</pre>
             MAX_ATT_CTL_ANGLE =
                   log( -true_airspeed ) * MAX_ATT_DAMPING_FACTOR ;
```

```
else
           MAX_ATT_CTL_ANGLE = MAX_ATT_CTL_ANGLE_STOP ;
     MAX_ATT_CTL_ANGLE = deg_to_rad( MAX_ATT_CTL_ANGLE );
#endif
      hover_aug_roll_integrator +=
                  HOVER_AUG_ROLL_I_GAIN * velocity_vector[X];
      hover_aug_roll_integrator =
                  limiter(-0.2,hover_aug_roll_integrator,0.2);
     hover_aug_roll_angle = HOVER_AUG_ROLL_P_GAIN * velocity_vector[X]
                  + hover_aug_roll_integrator;
     hover_aug_roll_angle = limiter (-MAX_ATT_CTL_ANGLE,
                              hover_aug_roll_angle,
                              MAX ATT_CTL_ANGLE);
        stab_aug_roll = set_roll_attitude (hover_aug_roll_angle);
      hover_aug_pitch_integrator +=
                  HOVER_AUG_PITCH_I_GAIN * velocity_vector[Y];
      hover_aug_pitch_integrator =
                  limiter(-0.2,hover_aug_pitch_integrator,0.2);
      hover_aug_pitch_angle = HOVER_AUG_PITCH_P_GAIN * velocity_vector[Y]
                  + hover_aug_pitch_integrator;
      hover_aug_pitch_angle = limiter (-MAX_ATT_CTL_ANGLE,
                                hover_aug_pitch_angle,
                                MAX ATT_CTL_ANGLE);
      stab_aug_pitch = set_pitch_attitude (hover_aug_pitch_angle);
      stab_aug_yaw_integrator -=
                  HOVER_AUG_YAW_I_GAIN * angular_velocity_vector[Z];
      if (stab_aug_yaw_integrator > 0.5) stab_aug_yaw_integrator = 0.5;
      if (stab_aug_yaw_integrator < -0.5) stab_aug_yaw_integrator = -0.5;
                     = - HOVER_AUG_YAW_P_GAIN * angular_velocity_vector[Z] +
      stab_aug_yaw
                        stab_aug_yaw_integrator;
      stab_aug_climb_integrator -=
                  HOVER_AUG_CLIMB_I_GAIN * velocity_vector[Z];
      if (stab_aug_climb_integrator > 0.2) stab_aug_climb_integrator = 0.2;
      if (stab_aug_climb_integrator < -0.2) stab_aug_climb_integrator = -0.2;
      stab_aug_climb = - HOVER_AUG_CLIMB_P_GAIN * velocity_vector[Z] +
                         stab aug_climb_integrator;
      stab_aug_yaw = limiter (-MAX_STAB_AUG_YAW_CLIMB_CONTROL,
                          stab_aug_yaw,
                          MAX_STAB_AUG_YAW_CLIMB_CONTROL);
      stab_aug_climb = limiter (-MAX_STAB_AUG_YAW_CLIMB_CONTROL,
                          stab_aug_climb,
                          MAX_STAB_AUG_YAW_CLIMB_CONTROL);
     }
     else
       stab_aug_roll = 0.0;
       stab aug_pitch = 0.0;
```

```
stab_aug_yaw = 0.0;
     stab_aug_climb = 0.0;
     pitch_damping = PITCH_RATE_DAMPING_GAIN; /* jwc 8/90 */
     roll_damping = ROLL_RATE_DAMPING_GAIN;
#ifdef notdef
                                          /* added 8/31/89 (jwc) */
     hover_aug_roll_integrator = 0.0;
     hover_aug_pitch_integrator = 0.0;
#endif
   controller_cyclic_roll = cyclic_roll + stab_aug_roll;
   controller_cyclic_pitch = cyclic_pitch + stab_aug_pitch;
    controller_tail_rotor = pedal + stab_aug_yaw;
    controller_collective = collective + stab_aug_climb;
}
static void send_aero_data_to_displays()
    if (velocity_vector[Y] > 0.0)
        meter_air_speed_set(true_airspeed);
    else
      meter_air_speed_set (0.0);
    meter_altitude_set(altitude);
    meter_vertical_speed_set(vertical_speed);
}
void aerodyn_simul()
    get_aircraft_kinematic_state();
    compute_flight_parameters();
    compute_stab_augmentation_gains();
    compute_rotor_loads();
    compute_engine_torque();
    compute rotor_forces_and_moments();
    compute_lift_drag_coefficients();
    compute_lift_drag_forces();
    compute_body_damping_forces_and_moments();
    transform_lift_drag_forces_to_body_coordinates();
    generate_gravity_body_force();
    interact_with_ground();
    sum_body_forces_and_moments_about_ac();
    send_to_dynamics_kinematics();
    send_aero_data_to_displays(); Must call if not calling orientation_calc */
    vehicle_update();
}
REAL aerodyn_get_true_airspeed()
    return (true_airspeed);
void aerodyn_set_hover_hold_on ()
```

```
hover_hold_state = ON;
}
void aerodyn_set_hover_hold_off()
   hover_hold_state = OFF;
   hover_hold_turned_on = FALSE;
   level_view = TRUE;
}
void aerodyn_toggle_hover_hold()
    if (hover_hold_state == OFF)
     hover_hold_state = ON;
    else
     hover_hold_state = OFF;
     hover_hold_turned_on = FALSE;
}
void forces_init ()
    aerodyn_init();
}
 * The following stuff is for the simplified dynamics model. The model is *
 * a modification of the aerodynamics model Warren wrote for the SAF.
 * Global variables defined for the real aerodynamics are reused here to
 * allow overlap in generic routines for operations such as control inputs,*
 * init, etc. - CJC
 ********************
#define MAX_HELICOPTER_POWER aero_simple[ 0]
                             aero_simple[ 1]
#define MAX_HH
/* constants for tweaking */
#define H_K1
                             aero_simple[ 2]
                             aero_simple[ 3]
#define H_K2
/* as increase drag coefficients, helicopter slows down faster */
#define H_K7
                             aero_simple[ 4]
                             aero_simple[ 5]
#define H_K8
                             aero_simple[ 6]
#define H_KP
                             aero_simple[ 7]
#define H_KPR
                             aero_simple[ 8]
#define H_KY
                             aero_simple[ 9]
#define H_KH
#define H_CHH
                             aero_simple[10]
#define H_CL
                             aero_simple[11]
void aerodyn_simple_simul ()
{
```

```
register int i;
 register REAL *vec_ptr;
 register REAL *res_ptr;
 register REAL *cur_ptr;
 register REAL *des_ptr;
REAL *drag_ptr;
REAL power;
REAL coll_factor;
REAL lift_factor;
VECTOR orient_vec;
VECTOR angular_accel;
VECTOR hover_hold_additions;
REAL euler[3];
                    /* euler angles */
VECTOR gravity_vector;
                          /* in body coordinates */
T_MAT_PTR C_mat;
                   /* direction cosine matrix */
get_aircraft_kinematic_state ();
generate_gravity_body_force();
compute_rotor_loads();
compute_engine_torque();
if (hover_hold_state == ON)
  hover_hold_additions[0] = min(velocity_vector[1] * H_KH,MAX_HH);
  hover_hold_additions[0] = max(hover_hold_additions[0],-MAX_HH);
  hover_hold_additions[1] = min(- velocity_vector[0] * H_KH,MAX_HH);
  hover_hold_additions[1] = max(hover_hold_additions[1],-MAX_HH);
  hover_hold_additions[2] = - velocity_vector[2] * H_KH * H_CHH;
}
else
  hover_hold_additions[0] = 0;
  hover_hold_additions[1] = 0;
  hover_hold_additions[2] = 0;
lift_factor = velocity_vector[1] * velocity_vector[1] * H_CL *
                   - cyclic_pitch;
/** original comment from SAF code **/
/* may want to put in power limit per unit time ... */
coll_factor = max(0.0,collective - 0.3);
power = H_KP * coll_factor + hover_hold_additions[2];
power += gross_weight * collective/(H_K2+collective) * 1.25;
power = min (MAX_HELICOPTER_POWER, power);
power = max (0.0, power);
if (fuel_level_empty ())
    power = 0.0;
/* Calculate the torque required to achieve the desired orientation */
/* orientation vector is [pitch element, roll element, yaw element] */
orient_vec[0] = H_KPR * - cyclic_pitch + hover_hold_additions[0];
```

orient\_vec[1] = H\_KPR \* cyclic\_roll + hover\_hold\_additions[1];

```
/** yaw element = current_yaw (heading) + rudder (pedals) * K **/
    orient_vec[2] = kinematics_get_yaw () + sign(pedal) * pedal
                      * pedal * H_KY;
    res_ptr = moment_body;
    des_ptr = orient_vec;
    C_mat = kinematics_get_w_to_h (veh_kinematics);
    euler[0] = atan2 (-gravity_dir_vector[Y], -gravity_dir_vector[Z]);
    euler[1] = - atan2 (-gravity_dir_vector[X], -gravity_dir_vector[Z]);
    euler[2] = kinematics_get_yaw ();
    cur_ptr = euler;
       First, compute the angular velocity necessary to achieve the */
    /* desired orientation in exactly one tick. (delta theta/ delta T) */
    /* Then get the angular acceleration needed to get to that velocity */
    /* In one tick.*/
    for (i = X; i \le Z; ++i)
     vec_ptr[i] = ((des_ptr[i] - cur_ptr[i]) / DELTA_T / H_K1);
     angular_accel[i] = (vec_ptr[i] - angular_velocity_vector[i])
                           / DELTA_T;
     res_ptr[i] = MOMENT_OF_INERTIA_X * angular_accel[i];
    }
   res_ptr[X] += lift_factor; /* this should add some torque for turns */
   /* compute force vector */
   res_ptr = force_body;
   cur_ptr = velocity_vector;
  vec_ptr = euler;
   drag_ptr = drag_force; /* drag_body or drag_force */
   drag_ptr[X] = square(cur_ptr[X]) * H_K8;
   drag_ptr[Y] = square(cur_ptr[Y]) * H_K7;
   drag_ptr[Z] = square(cur_ptr[Z]) * H_K8;
   res_ptr[X] = (sin(vec_ptr[Y]) * power) - (sign(cur_ptr[X]) * drag_ptr[X]);
   res_ptr[Y] = -(sin(vec_ptr[X]) * power) - (sign(cur_ptr[Y]) * drag_ptr[Y]);
   res_ptr[Z] = C_mat[2][2] * power;
   res_ptr[Z] -= sign(cur_ptr[Z]) * drag_ptr[Z];
   res_ptr[Z] += lift_factor; /* this should add some force for lift */
   vec_add (force_body, ground_force, force_body);
   vec_add (force_body, gravity_force_body, force_body);
   interact_with_ground();
   vec_add (force_body, force_ground_effect, force_body);
   vec_add (moment_body, ground_torque, moment_body);
   send_to_dynamics_kinematics ();
   vehicle_update ();
/*********************************
```

```
st The following is for the simplified model incorporating the stealth st
* dynamics. In this model, the cyclic changes the desired velocity
*************************
#define H_FWD_MUL aero_stealth[ 0]
#define H_SIDE_MUL
                       aero_stealth[ 1]
                       aero_stealth[ 2]
#define H_COLL_MUL
                       aero_stealth[ 3]
#define MAX_TORQUE
#define MAX_FORCE aero_stealth[ 4]
                       aero_stealth[ 5]
#define MASS
                       aero_stealth[ 6]
#define INERTIA
#define DEAD_ZONE aero_stealth[ 7]
/* use for gravity frame matrix. eliminate all pitch and roll
 * start with identity. substitute cos (yaw) for last term.
 */
static T_MATRIX level = {{1.0, 0.0, 0.0},
                  {0.0, 1.0, 0.0},
                  {0.0, 0.0, 1.0}};
void aerodyn_stealth_simul ()
    VECTOR desired_rot_vel;
    VECTOR desired_lin_vel;
   REAL adj_collective; /* collective value adjusted for dead zone and
                     for -1 to 1 range */
    adj_collective = (collective - 0.5) * 2.0; /* change to -1 to 1 */
    if (aerodyn_debug)
       timed_printf ("adj_collective = %.31f\n", adj_collective);
    if (allow_takeoff)
      if (adj_collective > 0.0)
          allow_takeoff = FALSE;
      }
      else
          adj_collective = 0.0;
      }
    get_aircraft_kinematic_state ();
    compute_rotor_loads();
    compute_engine_torque();
/* update desired velocity */
    desired_lin_vel[Z] = adj_collective * adj_collective *
          sign (adj_collective ) * H_COLL_MUL;
    if (hover_hold_state == ON)
```

```
{ /* no linear velocity in X,Y, only pitch */
     desired_lin_vel[X] = desired_lin_vel[Y] = 0.0;
     desired_rot_vel[X] = -cyclic_pitch * cyclic_pitch * sign(cyclic_pitch);
     desired_rot_vel[Y] = 0.0;
   }
   else
     if (level_view)/* when not in pitch mode, level view */
            vehicle_set_orientation_matrix (level); /* identity matrix */
            vehicle_set_orientation (kinematics_get_heading());
         level_view = FALSE;
      }
        desired_lin_vel[X] = cyclic_roll * cyclic_roll * sign (cyclic_roll)
              * H_SIDE_MUL;
        desired_lin_vel[Y] = cyclic_pitch * cyclic_pitch * sign (cyclic_pitch)
              * H_FWD_MUL;
        desired_rot_vel[X] = desired_rot_vel[Y] = 0.0;
#ifdef notdef
        desired_lin_vel[X] = cyclic_roll * cyclic_roll * sign (cyclic_roll)
              * H_SIDE_MUL;
        desired_lin_vel[Y] = cyclic_pitch * cyclic_pitch * sign (cyclic_pitch)
              * H_FWD_MUL;
        desired_rot_vel[X] = desired_rot_vel[Y] = 0.0;
#endif
   desired_rot_vel[Z] = pedal * pedal * sign(pedal);
    /* controller_forces */
    force_body[X] = (desired_lin_vel[X] - velocity_vector[X])
                     * MASS/DELTA_T;
    force_body[Y] = (desired_lin_vel[Y] - velocity_vector[Y])
                     * MASS/DELTA_T;
    force_body[Z] = (desired_lin_vel[Z] - velocity_vector[Z])
                     * MASS/DELTA_T;
    force_body[X] = min (MAX_FORCE, force_body[X]);
    force_body[Y] = min (MAX_FORCE, force_body[Y]);
    force_body[Z] = min (MAX_FORCE, force_body[Z]);
    force_body[X] = max (-MAX_FORCE, force_body[X]);
    force_body[Y] = max (-MAX_FORCE, force_body[Y]);
    force_body[Z] = max (-MAX_FORCE, force_body[Z]);
    /* controller_torques */
    moment_body[X] = (desired_rot_vel[X] - angular_velocity_vector[X])
                      * INERTIA/DELTA_T;
    moment_body[Y] = (desired_rot_vel[Y] - angular_velocity_vector[Y])
                      * INERTIA/DELTA_T;
    moment_body[Z] = (desired_rot_vel[Z] - angular_velocity_vector[Z])
                      * INERTIA/DELTA_T;
```

```
moment_body[X] = min (MAX_TORQUE, moment_body[X]);
   moment_body[Y] = min (MAX_TORQUE, moment_body[Y]);
   moment_body[Z] = min (MAX_TORQUE, moment_body[Z]);
   moment_body[X] = max (-MAX_TORQUE, moment_body[X]);
   moment_body[Y] = max (-MAX_TORQUE, moment_body[Y]);
   moment_body[Z] = max (-MAX_TORQUE, moment_body[Z]);
   interact_with_ground();
   vec_add (force_body, ground_force, force_body);
   vec_add (force_body, gravity_force_body,force_body);
   vec_add (force_body, force_ground_effect, force_body);
   send_to_dynamics_kinematics ();
   vehicle_update ();
}
/*************************
 * for tweaking purposes, use parameter file for constants
aerodyn_read_simple_constants (fn)
char *fn;
   char *strtok ();
   FILE *fp;
   char s[80];
   if ((fp = FOPEN (fn, "r")) == NULL)
     printf ("no tweakable constants file; using defaults\n", fn);
     return (-1);
   else
     printf ("Reading tweakable constants file: %s\n", fn);
   while (FGETS (s, 80, fp) != NULL)
      char *str;
      switch (s[0]) /* check for comments or blank lines */
        {
        case '#':
        case ' ':
        case '\n':
        case '\t':
          continue;
      str = strtok (s, " \t");
      if (strcmp (str, "H_K1") == 0)
          sscanf (strtok (0, " \t"), "%lf", &H_K1);
          continue;
```

```
}
if (strcmp (str, "H_K2") == 0)
    sscanf (strtok (0, " \t"), "%lf", &H_K2);
    continue;
}
if (strcmp (str, "H_K7") == 0)
    sscanf (strtok (0, " \t"), "%lf", &H_K7);
    continue;
}
if (strcmp (str, "H_K8") == 0)
    sscanf (strtok (0, " \t"), "%lf", &H_K8);
    continue;
}
if (strcmp (str, "H_KP") == 0)
    sscanf (strtok (0, " \t"), "%lf", &H_KP);
    continue;
}
if (strcmp (str, "H_KPR") == 0)
    sscanf (strtok (0, " \t"), "%lf", &H_KPR);
    continue;
if (strcmp (str, "H_KY") == 0)
    sscanf (strtok (0, " \t"), "%lf", &H_KY);
    continue;
}
if (strcmp (str, "H_KH") == 0)
    sscanf (strtok (0, " \t"), "%lf", &H_KH);
    continue;
}
if (strcmp (str, "H_FWD_MUL") == 0)
    sscanf (strtok (0, " \t"), "%lf", &H_FWD_MUL);
    continue;
}
if (strcmp (str, "H_COLL_MUL") == 0)
    sscanf (strtok (0, " \t"), "%lf", &H_COLL_MUL);
    continue;
}
```

## APPENDIX B - rwa\_aerodyn.c

if (strcmp (str, "H\_CHH") == 0)

```
sscanf (strtok (0, " \t"), "%lf", &H_CHH);
     continue;
 if (strcmp (str, "H_CL") == 0)
     sscanf (strtok (0, " \t"), "%lf", &H_CL);
     continue;
 }
 if (strcmp (str, "MAX_FORCE") == 0)
     sscanf (strtok (0, " \t"), "%lf", &MAX_FORCE);
     continue;
  }
  if (strcmp (str, "MAX_TORQUE") == 0)
      sscanf (strtok (0, " \t"), "%lf", &MAX_TORQUE);
      continue;
  }
  if (strcmp (str, "MASS") == 0)
      sscanf (strtok (0, " \t"), "%lf", &MASS);
      continue;
  }
  if (strcmp (str, "INERTIA") == 0)
      sscanf (strtok (0, " \t"), "%lf", &INERTIA);
      continue;
  }
  if (strcmp (str, "H_SIDE_MUL") == 0)
      sscanf (strtok (0, " \t"), "%lf", &H_SIDE_MUL);
      continue;
  }
  if (strcmp (str, "DEAD_ZONE") == 0)
      sscanf (strtok (0, " \t"), "%lf", &DEAD_ZONE);
      continue;
   /* if got here -- mistake */
  printf ("ERROR: Unknown constant %s in %s\n", str, fn);
FCLOSE (fp);
printf ("done reading constants file\n");
aerodyn_dump_simple_constants ();*/
return (1);
```

#### APPENDIX B - rwa\_aerodyn.c

```
}
aerodyn_dump_control_inputs ()
   printf ("collective = %.21f\tcyclic_roll = %.21f\tcyclic_pitch = %.21f\n",
          collective, cyclic_roll, cyclic_pitch);
    printf ("pedal = %.21f\n", pedal);
    aerodyn_debug = aerodyn_debug ? 0 : 1;
    printf ("aerodyn_debug is %s\n", aerodyn_debug ? "on" : "off");
}
aerodyn_dump_simple_constants ()
    printf ("Aerodyn simple constants:\n");
    printf ("\tH_K1:\t%.21f\n", H_K1);
    printf ("\tH_K2:\t%.21f\n", H_K2);
    printf ("\tH_K7:\t%.21f\n", H_K7);
    printf ("\tH_K8:\t%.21f\n", H_K8);
    printf ("\tH_KP:\t%.21f\n", H_KP);
    printf ("\tH_KPR:\t%.21f\n", H_KPR);
    printf ("\tH_KY:\t%.21f\n", H_KY);
    printf ("\tH_KH:\t%.21f\n", H_KH);
    printf ("\tH_FWD_MUL:\t%.21f\n", H_FWD_MUL);
    printf ("\tH_SIDE_MUL:\t%.21f\n", H_SIDE_MUL);
    printf ("\tH_COLL_MUL:\t%.21f\n", H_COLL_MUL);
    printf ("\tH_CHH:\t%.21f\n", H_CHH);
    printf ("\tH_CL:\t%.21f\n", H_CL);
    printf ("\tMAX_FORCE:\t%.21f\n", MAX_FORCE);
    printf ("\tMAX_TORQUE:\t%.21f\n", MAX_TORQUE);
    printf ("\tMASS:\t%.21f\n", MASS);
    printf ("\tINERTIA:\t%.21f\n", INERTIA);
    printf ("\tDEAD_ZONE:\t%.21f\n", DEAD_ZONE);
}
set_selected_model (model)
int model;
    switch (model)
      case COMPLEX_MODEL:
      printf ("switching to complex model, logarithmic collective\n");
      funny_little_kludge = 1;/* logarithmic collective */
      selected_model = model;
      break;
      case SIMPLE_MODEL:
      printf ("switching to simple model, linear collective\n");
      funny_little_kludge = 0;/* linear collective */
      selected_model = model;
      break;
      case STEALTH_MODEL:
      printf ("switching to stealth model, linear collective\n");
      funny_little_kludge = 0;/* linear collective */
      selected_model = model;
      break:
      default:
```

## APPENDIX B - rwa\_aerodyn.c

```
printf ("invalid selected model %d\n", model);
     printf ("using default complex model\n");
     selected_model = COMPLEX_MODEL;
     break;
   }
}
get_selected_model ()
    return (selected_model);
}
indicate_selected_model (model)
int model;
    switch (model)
      case COMPLEX_MODEL: -
      printf ("using complex model\n");
      break;
      case SIMPLE_MODEL:
      printf ("using simple model\n");
      break;
      case STEALTH_MODEL:
      printf ("using stealth model\n");
        allow_takeoff = TRUE;
      break;
      default:
      printf ("invalid selected model %d\n", model);
      printf ("using default complex model\n");
      break;
    }
}
set_takeoff_status (status)
int status;
    allow_takeoff = status;
 }
```

# Appendix C - Source code listing for rwa\_engine.c.

The following appendix contains the source code listing for rwa\_engine.c for convenience in document maintenance and understanding of the CSU.

```
/* $Header: /a3/adst-cm/RWA/simnet/vehicle/rwa/src/RCS/rwa_engine.c,v 1.5 1992/1
2/21 22:15:59 cm-adst Exp $ */
* $Log: rwa_engine.c,v $
* Revision 1.5 1992/12/21 22:15:59 cm-adst
* R. Branson's flight changes. These changes will become
 * BDS-D 1.1.1. This change was turned over by C. Swanson.
 * Revision 1.1 1992/10/07 19:00:23 cm-adst
 * Initial Version
static char RCS_ID[] = "$Header: /a3/adst-cm/RWA/simnet/vehicle/rwa/src/RCS/rwa_
engine.c,v 1.5 1992/12/21 22:15:59 cm-adst Exp $";
/****************************
* Revisions:
                                                        SP/CR Number
                                  Title
                       Author
              Date
     Version
                                  Data File Initiali-
              10/09/92 R. Branson
    1.2
                                     zation
                                  Data filenames changed
              10/16/92 R. Branson
    1.3
                                     to eight characters
                                  Added pathname to data
              10/30/92 R. Branson
    1.4
                                     directory
                                    Increased the size of the
    1.5 01/19/93 P.Desmeules
                                    fgets to make sure the
                                    whole line is read in.
        Description of Modification
       SP/CR No.
                     Hard coded defines changed to array elements.
                     Engine data array added.
                     Engine initialization data array added.
                     Engine status data array added.
                     Added file for engine data, engine initialization
                        data, and engine status data to the "engine_init"
                        function
                     Added "/simnet/data/" to each data file pathname.
        ******************************
 /******************************
               rwa_engine.c
  * FILE:
              James Chung
  * AUTHOR:
```

```
* MAINTAINER: James Chung
                4/19/89 james: Creation
* HISTORY:
* Copyright (c) 1989 BBN Systems and Technologies Corporation
* All rights reserved.
* Interim engine model for the generic rotary-wing aircraft
* with power characteristics similar to the General
* T700-GE-701 turboshaft engine. The T700 is rated at a
* maximum continuous power of 1510 shp at sea-level.
* Two (2) T700s power the AH-64 Apache attack helicopter.
#include "stdio.h"
#include "math.h"
#include "sim_dfns.h"
#include "sim_macros.h"
#include "sim_types.h"
#include "libsound.h"
#include "rwa_soun_dfn.h"
#include "rwa_meter.h"
#include "rwa_cntrl.h"
#include "libmun.h"
#include "failure.h"
#include "libfail.h"
* Debug macro
#ifdef FILEDBG
#define P(a)
#else
#define P(a)
#endif
/* Once the engine or transmission has been damaged, there is a chance that
   the engine/transmission will seize due to too many particle fragments
   accumulating in the respective oil system. These are "secondary" events.
                                       pjm */
                             12-10-90
#define DO_CFAIL TRUE /* do combat damage simulation */
#define DO_SFAIL TRUE /* do stochastic failure simulation */
static REAL engine_data[20] = {
                              0.05, 1031.6,
                                                    25.0,
      1030.55, 0.05,
                                                    34.0,
                             0.16438, 2.130,
                 1200.0,
         1.2,
                          153.8461539, 0.0,
                                                    0.0,
                100.0,
         7.0,
                  0.0, 133.84013
                                          0.0,
                                                      0.0
           0.0,
         } ;
static REAL engine_init_data[10] = {
                                                    0.0,
                                         0.0,
         0.0, 0.0, 0.0,
                                                     0.0
                                         0.0,
                              0.0,
         1.0,
                    0.0,
```

```
} ;
static int engine_stat_data[10] = {
                                           1,
                               1,
                    0,
         0,
                                           0,
                               0,
                    0,
         0,
         } ;
                                          engine_data[ 0]
#define GOVERNOR_ENGINE_SPEED_SETTING
                                     engine_data[ 1]
#define GOVERNOR_P_GAIN
                                     engine_data[ 2]
#define GOVERNOR_I_GAIN
                                     engine_data[ 3]
#define MAX_ENGINE_TORQUE
                                           engine_data[ 4]
#define MIN_ENGINE_LOAD_TORQUE
                                     engine_data[ 5]
#define MAX_ENGINE_PERCENT_POWER
                                           engine_data[ 6]
#define ENGINE_TORQUE_INTERCEPT
                                     engine_data[ 7]
#define ENGINE_TORQUE_SLOPE
                                     engine_data[8]
#define NOSE_GEARBOX_RATIO
                                     engine_data[ 9]
#define MAIN_ROTOR_GEAR_RATIO
                                     engine_data[10]
#define TAIL_ROTOR_GEAR_RATIO
                                     engine_data[11]
#define POWERTRAIN_INERTIA
                                     engine_data[12]
#define MAX_FUELFLOW
/* (seconds/tick) / (seconds/hour) = (hours/tick) */
#define HOURS_PER_TICK ( DELTA_T / 3600.0 )
static REAL hours_of_flight;
static int minutes_of_flight, old_minutes_of_flight;
static BOOLEAN engine_is_damaged, transmission_is_damaged;
/***** engine noise stuff *****/
#define ORIGINAL 0
#define BOTH_DISABLED
#define CHANGE_ROTOR
                         3
#define CHANGE_ENGINE
                         4
#define CHANGE_BOTH
static int engine_sound_type = CHANGE_BOTH;
static int engine_oscillation[2], rotor_oscillation[2];
                               105
 #define MIN_ROTOR_SOUND
                               120
 #define MAX_ROTOR_SOUND
                                (MAX_ROTOR_SOUND - MIN_ROTOR_SOUND)
 #define ROTOR_SOUND_RANGE
                                95
 #define MIN_TURBINE_SOUND
                                126
 #define MAX_TURBINE_SOUND
                                (MAX_TURBINE_SOUND - MIN_TURBINE_SOUND)
 #define TURBINE_SOUND_RANGE
 static REAL turbine_speed;
                                /* Nose gearbox output shaft */
 static REAL engine_speed;
 static REAL engine_load_torque;
 static REAL engine_percent_torque;
 static REAL engine_drive_torque;
 static REAL main_rotor_shaft_speed;
 static REAL main_rotor_drive_torque;
 static REAL tail_rotor_shaft_speed;
 static REAL tail_rotor_drive_torque;
 static REAL powertrain_percent_shaft_speed;
 static REAL last_percent_shaft_speed;
 static REAL last_percent_torque;
```

```
static REAL fuel_flow;
static REAL engine_power;
static REAL integrator_gain;
static REAL gov_p_gain;
static REAL gov_i_gain;
static int number_of_engines; /* Working */
static int engine_status;
/* Flag used to determine if the engine is starting. Sounds for the engine
   and rotors are more "realistic." Starting engine speed is 0 instead of
   GOVERNOR_ENGINE_SPEED_SETTING, and since engine_power then maxes out
   (causes "torque" to flash) a check is done and temporarily forces the
   torque percentage to be equal to 1.
                                                 Paul J. Metzger */
                              11-8-89
static int starting_engine;
        engine_simul (main_rotor_load, tail_rotor_load, altitude)
void
        main_rotor_load, tail_rotor_load, altitude;
REAL
            tail_rotor_engine_load;
    REAL
            main_rotor_engine_load;
    REAL
            temp_percent;
    REAL
            temp_sound;
    int
    main_rotor_engine_load = main_rotor_load / MAIN_ROTOR_GEAR_RATIO;
    tail_rotor_engine_load = tail_rotor_load / TAIL_ROTOR_GEAR_RATIO;
    engine_load_torque = main_rotor_engine_load + tail_rotor_engine_load;
    if (engine_load_torque < MIN_ENGINE_LOAD_TORQUE)</pre>
      engine_load_torque = MIN_ENGINE_LOAD_TORQUE;
    engine_power = gov_p_gain *
      (GOVERNOR_ENGINE_SPEED_SETTING - engine_speed);
    if (engine_status == WORKING)
      integrator_gain += gov_i_gain *
           (GOVERNOR_ENGINE_SPEED_SETTING - engine_speed);
      if (integrator_gain > 0.5)
          integrator_gain = 0.5;
      else if (integrator_gain < -0.5)
          integrator_gain = -0.5;
      engine_power += integrator_gain;
    }
                         /* Damaged */
    else
       integrator_gain = 0.0;
       if (engine_power > 0.7)
           engine_power = 0.7;
    }
    if (engine_power > MAX_ENGINE_PERCENT_POWER)
       engine_power = MAX_ENGINE_PERCENT_POWER;
```

```
if (engine_power < 0.0)
 engine_power = 0.0;
if (fuel_level_empty ()) /* Out of gas */
  engine_power = 0.0;
 engine_speed = 0.0;
engine_drive_torque = engine_power * number_of_engines *
  (ENGINE_TORQUE_INTERCEPT - ENGINE_TORQUE_SLOPE * engine_speed);
engine_percent_torque = engine_drive_torque /
  (MAX_ENGINE_TORQUE * number_of_engines);
if (engine_status == WORKING)
  engine_speed += (engine_drive_torque - engine_load_torque)
      / POWERTRAIN_INERTIA;
if (engine_speed < 0.0)
  engine_speed = 0.0;
turbine_speed = engine_speed * NOSE_GEARBOX_RATIO;
main_rotor_shaft_speed = engine_speed / MAIN_ROTOR_GEAR_RATIO;
tail_rotor_shaft_speed = engine_speed / TAIL_ROTOR_GEAR_RATIO;
powertrain_percent_shaft_speed = engine_speed /
  GOVERNOR_ENGINE_SPEED_SETTING;
tail_rotor_drive_torque = tail_rotor_load; /* Always have tail rotor */
main_rotor_drive_torque = (engine_drive_torque - tail_rotor_engine_load)
  * MAIN_ROTOR_GEAR_RATIO;
if (main_rotor_drive_torque < 0.0)</pre>
  main_rotor_drive_torque = 0.0;
fuel_flow = engine_percent_torque * MAX_FUELFLOW;
if (engine_status == BROKEN)/* crippled condition */
  sound_stop_cont_sound (SOUND_OF_STOP_ENGINE, SOUND_OF_VARY_ENGINE);
  sound_stop_cont_sound (SOUND_OF_STOP_ROTOR, SOUND_OF_VARY_ROTOR);
                         /* fuel leak */
  fuel_flow *= 50.0;
if (starting_engine)
  if (engine_percent_torque - .01 < .0001) /* within a delta */
       starting_engine = FALSE;
   else
       engine_percent_torque = .01;
 fuel_used_by_engine (fuel_flow / 3600.0 * DELTA_T);
 meter_torque_set (engine_percent_torque);
```

```
meter_rpm_set (powertrain_percent_shaft_speed);
   hours_of_flight += HOURS_PER_TICK;
   minutes_of_flight = (int) (hours_of_flight * 60);
#if DO_SFAIL
   if (minutes_of_flight > old_minutes_of_flight)
     sfail_event_occurred (SFAIL_EVENT_MILEAGE);
      if (engine_is_damaged)
          sfail_event_occurred (SFAIL_SECONDARY_EVENT_ENGINE);
      if (transmission_is_damaged)
          sfail_event_occurred (SFAIL_SECONDARY_EVENT_TRANSMISSION);
      old_minutes_of_flight = minutes_of_flight;
    }
#endif
    if (!fuel_level_empty ())
      switch (engine_sound_type)
      case CHANGE_ENGINE:
          if (abs (powertrain_percent_shaft_speed
                 - last_percent_shaft_speed) > 0.025)
            /* rotor sounds depend on RPMs
             * (powertrain_percent_shaft_speed) */
            temp_percent = max (0.01, powertrain_percent_shaft_speed);
            sound_make_cont_sound (SOUND_OF_START_ROTOR, SOUND_OF_VARY_ROTOR,
                                SOUND_OF_STOP_ROTOR, temp_percent);
            last_percent_shaft_speed = powertrain_percent_shaft_speed;
           }
           if (abs (engine_percent_torque - last_percent_torque) > 0.025)
             /* engine sounds depend on torque (engine_percent_torque) */
             temp_percent = max (0.01, engine_percent_torque);
             sound_make_cont_sound (SOUND_OF_START_ENGINE, SOUND_OF_VARY_ENGINE,
                                SOUND_OF_STOP_ENGINE, temp_percent);
             last_percent_torque = engine_percent_torque;
           }
           break;
       case ORIGINAL:
           if (abs (powertrain_percent_shaft_speed
                  - last_percent_shaft_speed) > 0.025)
             /* rotor sounds depend on RPMS
              * (powertrain_percent_shaft_speed) */
             temp_percent = max (0.01, powertrain_percent_shaft_speed);
             sound_make_cont_sound (SOUND_OF_START_ROTOR, SOUND_OF_VARY_ROTOR,
                                 SOUND_OF_STOP_ROTOR, temp_percent);
             sound_make_cont_sound (SOUND_OF_START_ENGINE, SOUND_OF_VARY_ENGINE,
                                 SOUND_OF_STOP_ENGINE, temp_percent);
             last_percent_shaft_speed = powertrain_percent_shaft_speed;
           }
```

```
break;
case CHANGE_BOTH:
    /* Try the following, as per Perc's directions: vary both the
     * rotor and engine with torque, but have the rotor range be from
     * 105 to 120, and the turbine range from 95 to 126.
     * The rotor sound range is 15 points (120-105), so the % torque is
     * multiplied by 15, then added to an offset of 105.
     * The turbine sound range is 31 points (126-95), so the % torque is
     * multiplied by 31, then added to an offset of 105.
                               */
     * 11-17-90
                  PJM
    if (abs (engine_percent_torque - last_percent_torque) > 0.025)
      /* both sounds depend on torque */
      temp_sound = (int) (engine_percent_torque * ROTOR_SOUND_RANGE) +
          MIN_ROTOR_SOUND;
      if (temp_sound > MAX_ROTOR_SOUND)
          temp_sound = MAX_ROTOR_SOUND;
      /* We check to see if the sounds are oscillating. This */
      /* event occurs while at the extreme torque edges of */
      /* the hover hold mode, when we're trying to break */
                              2-15-91 PJM
      /* hold.
      if (temp_sound != rotor_oscillation[1])
            sound_make_arg_sound (SOUND_OF_VARY_ROTOR, temp_sound);
      rotor oscillation[1] = rotor_oscillation[0];
      rotor_oscillation[0] = temp_sound;
      temp_sound = (int) (engine_percent_torque *
          TURBINE_SOUND_RANGE) + MIN_TURBINE_SOUND;
      if (temp_sound > MAX_TURBINE_SOUND)
          temp_sound = MAX_TURBINE_SOUND;
      if (temp_sound != engine_oscillation[1])
            sound_make_arg_sound (SOUND_OF_VARY_ENGINE, temp_sound);
      engine_oscillation[1] = engine_oscillation[0];
      engine_oscillation[0] = temp_sound;
      last_percent_torque = engine_percent_torque;
    break;
case CHANGE_ROTOR:
    if (abs (engine_percent_torque - last_percent_torque) > 0.025)
       /* rotor sounds depend on torque */
      temp_sound = (int) (engine_percent_torque * ROTOR_SOUND_RANGE) +
          MIN_ROTOR_SOUND;
       if (temp_sound > MAX_ROTOR_SOUND)
```

```
temp_sound = MAX_ROTOR_SOUND;
           sound_make_arg_sound (SOUND_OF_VARY_ROTOR, temp_sound);
           sound_stop_cont_sound (SOUND_OF_STOP_ENGINE,
                               SOUND_OF_VARY_ENGINE);
           last_percent_torque = engine_percent_torque;
         break;
     case BOTH_DISABLED:
         sound_stop_cont_sound (SOUND_OF_STOP_ENGINE, SOUND_OF_VARY_ENGINE);
         sound_stop_cont_sound (SOUND_OF_STOP_ROTOR, SOUND_OF_VARY_ROTOR);
          break;
      }
    }
}
        engine_get_rotor_percent_shaft_speed ()
REAL
    return (powertrain_percent_shaft_speed);
}
        engine_damage_engine_oil ()
void
#if DO_CFAIL
    controls_start_failure_lamp_flashing (MASTER_CAUTION);
    controls_start_failure_lamp_flashing (ENGINE_FAILURE);
#endif
    engine_is_damaged = TRUE;
}
        engine_repair_engine_oil ()
void
#if DO CFAIL
    controls_failure_lamp_off (ENGINE_FAILURE);
    engine_is_damaged = FALSE;
#endif
}
        engine_break_engine ()
void
    engine_status = BROKEN;
    engine_speed = 0.0;
    number_of_engines = 1;
}
         engine_repair_engine ()
void
     engine_repair_engine_oil ();
     engine_status = WORKING;
    number_of_engines = 2;
 }
         engine_damage_transmission_filter ()
 void
 {
```

```
#if DO_SFAIL
   controls_start_failure_lamp_flashing (MASTER_CAUTION);
   controls_start_failure_lamp_flashing (TRANSMISSION_FAILURE);
    transmission_is_damaged = TRUE;
#endif
}
       engine_repair_transmission_filter ()
void
#if DO_SFAIL
    controls_failure_lamp_off (TRANSMISSION_FAILURE);
    transmission_is_damaged = FALSE;
#endif
}
        engine_break_transmission ()
void
#if DO_SFAIL
    engine_break_engine ();  /* engine has seized */
#endif
}
        engine_repair_transmission ()
void
#if DO_SFAIL
    engine_repair_transmission_filter ();
    engine_repair_engine ();
#endif
}
        engine_init ()
void
      int
            i;
            data_init;
      int
      float data_tmp;
      char descript[80];
      FILE *fp;
      P(printf("$$$$ RWA ENGINE file data $$$$\n"););
                                                                    */
/* DEFAULT DATA FOR rwa_engine.c READ FROM FILE
      fp = fopen("/simnet/data/rwa_engn.d", "r");
      if(fp==NULL){
             fprintf(stderr, "Cannot open /simnet/data/rwa_engn.d\n");
             exit();
      }
      rewind(fp);
             Read array data
       /*
       i=0;
      while(fscanf(fp, "%f", &data_tmp) != EOF){
             engine_data[i] = data_tmp;
```

```
fgets(descript, 80, fp);
          P(printf("engine_data(%3d) is%11.3f %s", i, engine_data[i],
                       descript););
           ++i;
    }
     fclose(fp);
                                                                  */
 END DEFAULT DATA FOR rwa_engine.c READ FROM FILE
 DEFAULT INITIALIZATION DATA FOR rwa_engine.c READ FROM FILE
                                                                        */
     fp = fopen("/simnet/data/rw_en_in.d","r");
    if(fp==NULL){
           fprintf(stderr, "Cannot open /simnet/data/rw_en_in.d\n");
           exit();
     }
     rewind(fp);
           Read array data
     i=0;
     while(fscanf(fp, "%f", &data_tmp) != EOF){
           engine_init_data[i] = data_tmp;
           fgets(descript, 80, fp);
           P(printf("engine_init_data(%3d) is%11.3f %s", i,
                   engine_init_data[i], descript););
           ++i;
     }
     fclose(fp);
   END DEFAULT INITIALIZATION DATA FOR rwa_engine.c READ FROM FILE
                                                                         */
   DEFAULT STATUS DATA FOR rwa_engine.c READ FROM FILE
     fp = fopen("/simnet/data/rw_en_st.d","r");
     if(fp==NULL){
           fprintf(stderr, "Cannot open /simnet/data/rw_en_st.d\n");
           exit();
     }
     rewind(fp);
           Read array data
     i=0;
     while(fscanf(fp, "%d", &data_init) != EOF){
            engine_stat_data[i] = data_init;
            fgets(descript, 80, fp);
            P(printf("engine_stat_data(%3d) is%11d %s", i,
                    engine_stat_data[i], descript););
            ++i;
      }
      fclose(fp);
                                                                    */
/* END DEFAULT STATUS DATA FOR rwa_engine.c READ FROM FILE
```

```
GOVERNOR_P_GAIN;
   gov_p_gain =
                               GOVERNOR_I_GAIN;
   gov_i_gain =
                               engine_init_data[ 0];
   engine_power =
                               engine_init_data[ 1];
   engine_percent_torque =
                               engine_init_data[ 2];
   engine_speed =
                               engine init_data[ 3];
   integrator_gain =
   last_percent_shaft_speed = engine_init_data[ 4];
                               engine_init_data[ 5];
   last_percent_torque =
                               engine_init_data[ 6];
   hours_of_flight =
                             engine_stat_data[ 0];
   minutes_of_flight =
                               engine_stat_data[ 1];
   old_minutes_of_flight =
                               engine_stat_data[ 2];
   engine_status =
                               engine_stat_data[ 3];
   starting_engine =
                               engine_stat_data[ 4];
   number_of_engines =
                               engine_stat_data[ 5];
    engine_is_damaged =
   transmission_is_damaged = engine_stat_data[ 6];
#if DO_CFAIL
    fail_init_failure (motiveOilLeak, engine_damage_engine_oil,
                 .engine_repair_engine_oil, NO_SELF_REPAIR, noncritKill);
    fail_init_failure (motiveEngineMajor, engine_break_engine,
                   engine_repair_engine, NO_SELF_REPAIR, mobilityKill);
#endif
#if DO_SFAIL
    fail_init_failure (motiveTransFluidFilter,
       engine_damage_transmission_filter, engine_repair_transmission_filter,
                   NO_SELF_REPAIR, noncritKill);
    fail_init_failure (motiveTransmissionMajor, engine_break_transmission,
              engine_repair_transmission, NO_SELF_REPAIR, mobilityKill);
#endif
}
void
        engine_debug_print ()
    printf ("rpm = %f\n rps = %f\n ps = %f\n etq = %f\n mrt = %f\n",
          powertrain_percent_shaft_speed, engine_speed,
          engine_power, engine_drive_torque, main_rotor_drive_torque);
}
        engine_get_speed ()
REAL
    return (engine_speed);
void
        engine_toggle_sound ()
    if ((engine_sound_type - 1) < ORIGINAL)</pre>
      engine_sound_type = CHANGE_BOTH;
      engine_sound_type--;
    switch (engine_sound_type)
```

```
case ORIGINAL:
     printf ("Rotor: RPM Engine: RPM\n");
     break;
   case CHANGE_ROTOR:
     printf ("Rotor: TORQUE Engine: DISABLED\n");
     break;
   case CHANGE_ENGINE:
                          Engine: TORQUE\n");
     printf ("Rotor: RPM
     break;
   case CHANGE_BOTH:
     printf ("Rotor: TORQUE ' Engine: TORQUE\n");
     break;
   case BOTH_DISABLED:
     printf ("Rotor: DISABLED Engine: DISABLED\n");
     break;
    }
}
       engine_get_hours_of_flight ()
REAL
    return (hours_of_flight);
} *
        engine_get_minutes_of_flight ()
int
    return (minutes_of_flight);
}
```

# Appendix D- Source code listing for rwa\_kinemat.c.

The following appendix contains the source code listing for rwa\_kinemat.c for convenience in document maintenance and understanding of the CSU.

```
/* $Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/rwa/src/RCS/rwa_kinemat.c,v
1.6 1993/01/28 23:33:00 cm-adst Exp $ */
 * $Log: rwa_kinemat.c,v $
 * Revision 1.6 1993/01/28 23:33:00 cm-adst
 * P. DesMeueles's changes for spcr 31
 * Revision 1.5 1992/12/21 22:16:49 cm-adst
 * R. Branson's flight changes. These changes will become
 * BDS-D 1.1.1. This change was turned over by C. Swanson.
 * Revision 1.1 1992/10/07 19:00:23 cm-adst
 * Initial Version
*/
static char RCS_ID[] = "$Header: /a3/adst-
cm/RWA/AIRNET/simnet/vehicle/rwa/src/RCS/rwa_kinemat.c,v 1.6 1993/01/28 23:33:00
cm-adst Exp $";
/****************************
  Revisions:
                                                            SP/CR Number
                              Author Title
                  Date
      Version
                  10/09/92 R. Branson Data File Initiali-
      1.2
                                       zation
                   10/16/92 R. Branson Data filenames changed
      1.3
                                        to eight characters
                 10/30/92 R. Branson Added pathname to data
     1.4
                                        directory
                   01/19/93 P.Desmeules Increased the size of the
                                                                  31
     1.5
                                       fgets to make sure the
                                       whole line is read in.
                                                                   85
                   03/04/93 P.Desmeules Fix value of
      1.5
                                       DISPLAY_SPEED_LIMIT
                      Description of Modification
        SP/CR No.
                       Hard coded defines changed to array element.
                       Kinemat data array added.
                       Kinemat initialization array added.
                       Added file read for kinemat data and kinemat initiali-
                          zation data to the "veh_spec_kinematics_init"
                          function.
                       Added "/simnet/data/" to each data file pathname.
  **************************
```

```
/**********************
               rwa_kinemat.c
* FILE:
               Bryant Collard
* AUTHOR:
* MAINTAINER: Bryant Collard
* PURPOSE: This file contains routines which process
                information generated in the dynamics and
                kinematics software to generate data needed
                specifically for the rotary wing aircraft.
               03/03/89 bryant: Creation
* HISTORY:
                05/15/89 james: Modified for RWA
* Copyright (c) 1989 BBN Systems and Technologies, Inc.
* All rights reserved.
****************************
#include "stdio.h"
#include "math.h"
#include "sim_types.h"
#include "sim_dfns.h"
#include "sim_macros.h"
#include "libmatrix.h"
#include "librotate.h"
#include "vehicle.h"
#include "std_atm.h"
* Debug macro
#ifdef FILEDBG
#define P(a)
#else
#define P(a)
#endif
                            kinemat_data[ 0]
#define GRAV_CONSTANT
                            kinemat_data[ 1]
#define SIN_AOA_LIMIT
                            kinemat_data[ 2]
#define COS_AOA_LIMIT
                            kinemat_data[ 3]
#define SIN_YAW_LIMIT
                            kinemat_data[ 4]
#define COS_YAW_LIMIT
                            kinemat_data[5]
#define DISPLAY_SPEED_LIMIT
static VECTOR pos_unit_vel;
static VECTOR neg_unit_vel;
static REAL sin_aoa;
static REAL cos_aoa;
static REAL sin_yaw;
static REAL cos_yaw;
```

```
static REAL altitude;
static REAL body_pitch;
static REAL body_pitch_offset;
static REAL velocity_pitch;
static REAL roll;
static REAL heading;
static REAL true_airspeed;
static REAL indicated_airspeed;
static REAL g_force;
static REAL vertical_speed;
static REAL *ang_vel;
static REAL *velocity_vector;
static VECTOR gravity;
static VECTOR norm_vel;
static T_MATRIX velocity_to_body;
* SPCR 85 - Fix the value of DISPLAY_SPEED_LIMIT (element 5) from 0.0 to 5.0
static REAL kinemat_data[20] = {
     9.81, 0.642787610, 0.766044443, 0.642787610, 0.766044443,
     3.0,
                                            0.0
                        0.0,
              0.0,
     0.0,
     } ;
static REAL kinemat_init_data[30] = {
                               0.0,
0.0,
                                           -1.0,
     0.0, 1.0, 0.0,
                                           1.0,
                       1.0,
             0.0,
     0.0,
                       0.0,
0.0,
1.0,
                                            0.0,
             0.0,
     0.0,
                                            0.0,
             0.0,
     0.0,
             0.0,
                                            1.0,
     0.0,
                                             0.0
                                  0.0,
                       0.0,
     0.0,
              0.0,
     } ;
/************************
 * ROUTINE: veh_spec_kinematics_init
 * PARAMETERS: none
 * RETURNS: none
 * PURPOSE: This routine initializes vehicle specific *
          kinematics parameters.
 *******************
void veh_spec_kinematics_init ()
/* DEFAULT DATA FOR rwa_kinemat.c READ FROM FILE */
     int i;
     float data_tmp;
     char descript[80];
FILE *fp;
     P(printf("$$$$ RWA KINEMATICS file data $$$$\n"););
```

```
fp = fopen("/simnet/data/rwa_kine.d","r");
     if(fp==NULL){
           fprintf(stderr, "Cannot open /simnet/data/rwa_kine.d\n");
           exit();
     }
     rewind(fp);
           Read array data */
     /*
     i=0;
     while(fscanf(fp,"%f", &data_tmp) != EOF){
           kinemat_data[i] = data_tmp;
            fgets(descript, 80, fp);
           P(printf("kinemat_data(%3d) is%11.3f %s", i, kinemat_data[i],
                        descript););
            ++i;
      }
      fclose(fp);
   END DEFAULT DATA FOR rwa_kinemat.c READ FROM FILE
/* DEFAULT INITIALIZATION DATA FOR rwa_kinemat.c READ FROM FILE
      fp = fopen("/simnet/data/rw_ki_in.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/rw_ki_in.d\n");
            exit();
      }
      rewind(fp);
      /*
            Read array data
      i=0;
      while(fscanf(fp, "%f", &data_tmp) != EOF){
            kinemat_init_data[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("kinemat_init_data(%3d) is%11.3f %s", i,
                        kinemat_init_data[i], descript););
            ++i;
      }
      fclose(fp);
                                                                           */
/* END DEFAULT INITIALIZATION DATA FOR rwa_kinemat.c READ FROM FILE
                         kinemat_init_data[ 1];
    pos_unit_vel[Y] =
                          kinemat_init_data[ 2];
    pos_unit_vel[Z] =
                          kinemat_init_data[ 3];
    neg_unit_vel(X) =
                          kinemat_init_data[ 4];
    neg_unit_vel[Y] =
                          kinemat_init_data[ 5];
    neg_unit_vel[Z] =
                          kinemat_init_data[ 6];
    sin_aoa =
```

```
kinemat_init_data[ 7];
   cos_aoa =
                        kinemat_init_data[ 8];
   sin_yaw =
                        kinemat_init_data[ 9];
   cos_yaw =
                        kinemat_init_data[10];
   altitude =
                        kinemat_init_data[11];
   body_pitch =
   body_pitch_offset = kinemat_init_data[12];
                        kinemat_init_data[13];
   velocity_pitch =
                        kinemat_init_data[14];
   roll =
                        kinemat_init_data[15];
   heading =
   heading = kinemat_init_data[15];
true_airspeed = kinemat_init_data[16];
   indicated_airspeed = kinemat_init_data[17];
                       kinemat_init_data[18];
   g_force =
                        kinemat_init_data[19];
   vertical_speed =
   ang_vel = vehicle_angular_velocity ();
   velocity_vector = vehicle_velocity();
                        kinemat_init_data[20];
   gravity[X] =
                        kinemat_init_data[21];
   gravity[Y] =
                       kinemat_init_data[22];
   gravity[Z] =
                      kinemat_init_data[23];
   norm_vel[X] =
                       kinemat_init_data[24];
   norm_vel[Y] =
                       kinemat_init_data[25];
   norm_vel[Z] =
   mat_ident (velocity_to_body);
       *****************
 * ROUTINE: veh_spec_kinematics_simul
 * PARAMETERS:
                 none
 * RETURNS: none
 * PURPOSE: This routine finds vehicle specific kinematics *
           parameters.
     *********
void veh_spec_kinematics_simul ()
   REAL *velocity;
    REAL temp, temp2;
    REAL *position;
   T_MAT_PTR body_to_world;
    position = rotate_get_loc (world (), hull ());
    altitude = position[Z];
    if (altitude < 0.0)
        altitude = 0.0;
      velocity = vehicle_velocity (); */
    velocity = velocity_vector;
    true_airspeed = sqrt (velocity[X] * velocity[X] + velocity[Y] * velocity[Y]
            + velocity[Z] * velocity[Z]);
    indicated_airspeed = true_airspeed * sqrt (air_density (altitude) /
            air_density(0.0));
    if (true_airspeed < E_MILLI)</pre>
        norm_vel[X] = 0.0;
        norm_vel[Y] = 1.0;
```

```
norm_vel[Z] = 0.0;
  }
  else
  {
      norm_vel[X] = velocity[X] / true_airspeed;
      norm_vel[Y] = velocity[Y] / true_airspeed;
      norm_vel[Z] = velocity[Z] / true_airspeed;
  if (norm_vel[Z] - 1.0 > -E_NANO)
      sin_aoa = -1.0;
      cos_aoa = 0.0;
       sin_yaw = 0.0;
      cos_yaw = 1.0;
  else if (norm_vel[Z] + 1.0 < E_NANO)
       sin_aoa = 1.0;
       cos_aoa = 0.0;
       sin_yaw = 0.0;
       cos_yaw = 1.0;
   }
   else
   {
       sin_aoa = -norm_vel[Z];
       cos_aoa = sqrt (norm_vel[X] * norm_vel[X] + norm_vel[Y] * norm_vel[Y]);
       sin_yaw = norm_vel[X] / cos_aoa;
       cos_yaw = norm_vel[Y] / cos_aoa;
   }
   if (sin_aoa > SIN_AOA_LIMIT)
       temp = COS_AOA_LIMIT;
       velocity_to_body[1][2] = -SIN_AOA_LIMIT;
   else if (sin_aoa < -SIN_AOA_LIMIT)</pre>
        temp = COS_AOA_LIMIT;
        velocity_to_body[1][2] = SIN_AOA_LIMIT;
    }
    else
    {
*/
        temp = cos_aoa;
        velocity_to_body[1][2] = -sin_aoa;
    if (cos_yaw < COS_YAW_LIMIT)
    {
        velocity_to_body[0][0] = COS_YAW_LIMIT;
        if (\sin_yaw > 0)
            velocity_to_body[0][1] = -SIN_YAW_LIMIT;
        else
            velocity_to_body[0][1] = SIN_YAW_LIMIT;
    }
```

```
else
*/
        velocity_to_body[0][0] = cos_yaw;
        velocity_to_body[0][1] = -sin_yaw;
* /
    velocity_to_body[0][2] = 0.0;
    velocity_to_body[1][0] = -velocity_to_body[0][1] * temp;
    velocity_to_body[1][1] = velocity_to_body[0][0] * temp;
    velocity_to_body[2][0] = velocity_to_body[1][2] * velocity_to_body[0][1];
    velocity_to_body[2][1] = -velocity_to_body[1][2] * velocity_to_body[0][0];
velocity_to_body[2][2] = velocity_to_body[1][1] * velocity_to_body[0][0] -
            velocity_to_body[1][0] * velocity_to_body[0][1];
    ang vel = vehicle_angular_velocity ();
    body_to_world = rotate_get_mat (hull (), world ());
    gravity[X] = body_to_world[0][2];
    gravity[Y] = body_to_world[1][2];
    gravity[Z] = body_to_world[2][2];
    g_force = gravity[Z] + (true_airspeed * ang_vel[X] / GRAV_CONSTANT);
 vertical_speed = vec_dot_prod (norm_vel, gravity);
    if (true_airspeed >= DISPLAY_SPEED_LIMIT)
        velocity_pitch = asin (vertical_speed);
    else
        velocity_pitch = 0.0;
    vertical_speed *= true_airspeed;
    body_pitch = asin (body_to_world[1][2]);
    gravity[X] = -gravity[X];
    gravity[Y] = -gravity[Y];
    gravity[Z] = -gravity[Z];
    temp = sqrt (body_to_world[1][0] * body_to_world[1][0] +
            body_to_world[1][1] * body_to_world[1][1]);
    if (temp < E_NANO)
    {
        roll = 0.0;
        heading = 0.0;
    }
    else
      temp2 = (body_to_world[0][0] * body_to_world[1][1] -
            body_to_world[0][1] * body_to_world[1][0]) / temp;
      if (temp2 > 1.0) temp2 = 1.0;
      roll = acos (temp2);
        if (body_to_world[1][1] * body_to_world[2][0] -
                 body_to_world[1][0] * body_to_world[2][1] < 0.0)
             roll = -roll;
        if (body_to_world[1][0] >= 0.0)
             heading = acos (body_to_world[1][1] / temp);
        else
             heading = acos (-body_to_world[1][1] / temp) + PI;
    }
/* NO METERS FOR NOW
    meter_q_force_set (g_force);
    meter_vertical_speed_set (vertical_speed);
```

```
if (true_airspeed >= DISPLAY_SPEED_LIMIT)
       meter_send_aero_data (rad_to_deg (body_pitch), rad_to_deg (roll),
              rad_to_deg (heading), asin (sin_aoa), asin (sin_yaw),
                indicated_airspeed, altitude, g_force);
    else
       meter_send_aero_data (0.0, 0.0,
                rad_to_deg (heading), 0.0, 0.0,
                indicated_airspeed, altitude, g_force);
*/
REAL kinematics_get_aoa ()
    return (asin (-velocity_to_body[1][2]));
}
REAL kinematics_get_yaw ()
    return (asin (-velocity_to_body[0][1]));
}
REAL kinematics_get_altitude ()
    return (altitude);
}
REAL kinematics_get_body_pitch ()
    return (body_pitch + body_pitch_offset);
}
REAL kinematics_get_velocity_pitch ()
    return (velocity_pitch);
REAL kinematics_get_roll ()
    return (roll);
}
REAL kinematics_get_heading ()
    return (heading);
}
REAL kinematics_get_true_airspeed ()
     return (true_airspeed);
 }
REAL kinematics_get_indicated_airspeed ()
     return (indicated_airspeed);
 }
```

```
REAL kinematics_get_g_force ()
    return (g_force);
REAL kinematics_get_vertical_speed ()
    return (vertical_speed);
}
REAL *kinematics_get_gravity_vector ()
    return (gravity);
REAL *kinematics_get_linear_velocity_vector()
    return (velocity_vector);
}
REAL *kinematics_get_normalized_velocity_vector ()
    if (true_airspeed > DISPLAY_SPEED_LIMIT)
        return (norm_vel);
    else if (norm_vel[Y] >= 0.0)
        return (pos_unit_vel);
    else
        return (neg_unit_vel);
}
REAL *kinematics_get_angular_velocity_vector ()
    return (ang_vel);
T_MAT_PTR kinematics_get_velocity_to_body ()
    return (velocity_to_body);
}
```

# Appendix E - Source code listing for miss\_adat.c.

The following appendix contains the source code listing for miss\_adat.c for convenience in document maintenance and understanding of the CSU.

```
/* $Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/libmissile/RCS/miss_ada
t.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $ */
/*
 * $Log: miss_adat.c,v $
 * Revision 1.4 1993/01/28 23:22:08 cm-adst
 * P.DesMeules changes for spcr 31
* Revision 1.3 1993/01/06 21:12:09 cm-adst
 * R.Branson's changes for weapons model.
 * Revision 1.1 1992/09/30 16:39:52 cm-adst
 * Initial Version
*/
static char RCS_ID[] = "$Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/li
bmissile/RCS/miss_adat.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $";
/*****************************
* Revisions:
                                                           SP/CR Number
                           Author
                                    Title
     Version
                 Date
                 10/23/92 R. Branson Data File Initiali-
     1.2
                                       zation
                 10/30/92 R. Branson Added pathname to data
     1.3
                                       directory
                 11/25/92 R. Branson Changed %i to %d
     1.4
                  01/19/93 P.Desmeules Increased the size of the
                                                                 31
     1.5
                                       fgets to make sure the
                                       whole line is read in.
       SP/CR No.
                     Description of Modification
                     Hard coded defines changed to array elements.
                     Characteristics/parameter data array added.
                     Engine initialization data array added.
                     Degree of polynomial data array added.
                      Added file reads for ADAT characteristics/
                        parameters, burn speed coefficients, coast speed
                        coefficients, burn turn coefficients, coast turn
                        coefficients, and temporal bias coefficients.
                      Added "/simnet/data/" to each data file pathname.
       *****************************
/**********************
```

```
miss_adat.c
* FILE:
             Bryant Collard
* AUTHOR:
* MAINTAINER: Bryant Collard
              This file contains routines which fly out a
* PURPOSE:
              missile with the characteristics of a ADAT
              missile.
             06/28/89 bryant: Creation
* HISTORY:
              08/06/90 bryant: NIU librva modifications.
* Copyright (c) 1989 BBN Systems and Technologies, Inc.
* All rights reserved.
*************************
#include "stdio.h"
#include "math.h"
#include "sim_types.h"
#include "sim_dfns.h"
#include "basic.h"
#include "mun_type.h"
#include "libmap.h"
#include "libmatrix.h"
#include "miss_adat.h"
#include "libmiss_dfn.h"
#include "libmiss_loc.h"
* Debug macro
#ifdef FILEDBG
#define P(a)
#else
#define P(a)
#endif
* Define missile characteristics.
/*/
                             adat_miss_char[ 0]
#define ADAT_BURNOUT_TIME
#define ADAT_MAX_FLIGHT_TIME adat_miss_char[ 1]
                             adat_miss_char[ 2]
#define INVEST_DIST_SQ
                             adat_miss_char[ 3]
#define HELO_FUZE_DIST_SQ
#define AIR_FUZE_DIST_SQ
                             adat_miss_char[ 4]
                           adat_miss_char[5]
#define ADAT_TEMP_BIAS_TIME
                             adat_miss_char[ 6]
#define CLOSE_RANGE
 * Define the states the _ADAT_MISSILE_ can be in.
```

```
/* No missile assigned. */
#define ADAT_FREE
                    0
                          /* Missile flying and guided. */
#define ADAT_GUIDE 1
                          /* Missile flying but unguided. */
#define ADAT_UNGUIDE 2
                         /* Missile flying against a close target. */
#define ADAT_CLOSE 3
                          /* Missile fired without cooling. */
                     4
#define ADAT_HOT
/*/
 * The following terms set the order of the polynomials used to determine
 * the speed or cosine of the maximum allowed turn rate of the missile
 * at any point in time.
/*/
                              adat_miss_poly_deg[ 0]
#define ADAT_BURN_SPEED_DEG
#define ADAT_COAST_SPEED_DEG adat_miss_poly_deg[ 1]
                              adat_miss_poly_deg[ 2]
#define ADAT_BURN_TURN_DEG
                              adat_miss_poly_deg[ 3]
#define ADAT_COAST_TURN_DEG
                              adat_miss_poly_deg[ 4]
#define ADAT_TEMP_BIAS_DEG
 * ADAT missile characteristic parameters initialized to default values.
/*/
static REAL adat_miss_char[10] =
             /* ticks (3.2 sec) */
    48.0,
             /* ticks (20.0 sec) */
   300.00,
             /* (300 m) ** 2 */
  90000.0,
             /* (7 m) ** 2 */
    49.0,
             /* (14 m) ** 2 */
    196.0,
             /* ticks (4.0 sec) */
     60.0,
             /* close range*/
   2200.0,
      0.0,
      0.0,
      0.0
 };
 /*/
 * The following are the default values of the degree of polynomials.
 /*/
 static int adat_miss_poly_deg[5] =
 {
              /* Speed before motor burnout. */
    2,
               /* Speed after motor burnout. */
    4,
               /* Cosine of max turn before burnout. */
               /* Cosine of max turn after burnout. */
               /* Temporial bias. */
    4
 };
  * Coefficients for the speed polynomial before motor burnout.
  static REAL adat_burn_speed_coeff[10] =
```

```
/* a_0 - m/tick */
   2.296,
                       /* a_1 - m/tick**2 */
   0.72990856,
                       /* a_2 - m/tick**3 */
   0.013310932,
   0.0,
   0.0,
   0.0,
   0.0,
   0.0,
   0.0,
   0.0
};
* Coefficients for the speed polynomial after motor burnout.
/*/
static REAL adat_coast_speed_coeff[10] =
                        /* a_0 - m/tick */
  105.52162,
                        /* a_1 - m/tick**2 */
   -1.0157285,
                        /* a_2 - m/tick**3 */
   5.6124330e-3,
                        /* a_3 - m/tick**4 */
   -1.6262608e-5,
                        /* a_4 - m/tick**5 */
   1.8991982e-8,
    0.0,
    0.0,
    0.0,
    0.0,
    0.0
};
 * Coefficients for the cosine of max turn polynomial before motor burnout.
static REAL adat_burn_turn_coeff[10] =
 {
                        /* a_0 - cos(rad)/tick */
     0.999993,
                        /* a_1 - cos(rad)/tick**2 */
   -6.2386917e-7,
                        /* a_2 - cos(rad)/tick**3 */
    1.6146426e-7,
                        /* a_3 - cos(rad)/tick**4 */
    -9.720142e-7,
     0.0,
     0.0,
     0.0,
     0.0,
     0.0,
     0.0
 };
  * Coefficients for the cosine of max turn polynomial after motor burnout.
 /*/
 static REAL adat_coast_turn_coeff[10] =
                         /* a_0 - cos(rad)/tick */
     0.99753111,
```

```
/* a_1 - cos(rad)/tick**2 */
   5.5817986e-5,
                       /* a_2 - cos(rad)/tick**3 */
   -5.1276276e-7,
                       /* a_3 - cos(rad)/tick**4 */
   2.2388593e-9,
                        /* a_4 - cos(rad)/tick**5 */
   -5.1964622e-12,
                        /* a_5 - cos(rad)/tick**6 */
   4.5499104e-15,
   0.0,
   0.0,
   0.0,
   0.0
};
* Coefficients for the temporial bias polynomial.
static REAL adat_temp_bias_coeff[10] =
                       /* a_0 - m */
    5.3105657e-2,
                       /* a_1 - m/tick */
    7.1795817e-2,
                       /* a_2 - m/tick**2 */
    1.8084646e-2,
                       /* a_3 - m/tick**3 */
   -6.0083762e-4,
                        /* a_4 - m/tick**4 */
 4.6761091e-6,
    0.0,
    0.0,
    0.0,
    0.0,
    0.0
};
 * The following arrays are used to give the missile the proper superelevation
 * at launch time. Two are required to deal with launches off either side
 * of the turret.
/*/
static T_MATRIX tube_C_sight_left;
static T_MATRIX tube_C_sight_right;
/*/
 * Memory for the missiles is declared in vehicle specific code. During
 * initialization; a pointer is assigned to this memory then some memory
 * issues are dealt with in this module.
/*/
static ADAT_MISSILE *adat_array; /* A pointer to missile memory. */
                                     /* The number of defined missiles. */
static int num_adats;
 * Declare static functions.
/*/
/* static void missile_adat_fly (); ** made external */
static void missile_adat_stop ();
```

```
* ROUTINE: missile_adat_init
* PARAMETERS: missile_array - A pointer to an array of
                          ADAT missiles defined in *
                           vehicle specific code.
           num_missiles - The number missiles defined in
                          _missile_array_.
 * RETURNS: none
 * PURPOSE: This routine copies the parameters into
           variables static to this module and initializes *
           the state of all the missiles. It also
           initializes the proximity fuze.
        *********
void missile_adat_init (missile_array, num_missiles)
ADAT_MISSILE missile_array[];
int num_missiles;
               /* A counter. */
      int
     REAL mag; /* Used to generate tube to sight matricies. */
           data_tmp_int;
      float data_tmp;
      char descript[80];
      FILE *fp;
      P(printf("$$$$ ADAT missile file data $$$$\n"););
/* DEFAULT CHARACTERISTICS DATA FOR miss_adat.c READ FROM FILE
                                                                       */
      fp = fopen("/simnet/data/ms_ad_ch.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_ad_ch.d\n");
            exit();
      }
      rewind(fp);
            Read array data */
      i=0;
      while(fscanf(fp,"%f", &data_tmp) != EOF){
            adat_miss_char[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("adat_miss_char(%3d) is%11.3f %s", i,
                  adat_miss_char[i], descript););
            ++i;
      }
       fclose(fp);
 /* END DEFAULT CHARACTERISTICS DATA FOR miss_adat.c READ FROM FILE
 /* DEFAULT BURN SPEED DATA FOR miss_adat.c READ FROM FILE
       fp = fopen("/simnet/data/ms_ad_bs.d", "r");
       if(fp==NULL){
```

```
fprintf(stderr, "Cannot open /simnet/data/ms_ad_bs.d\n");
        exit();
 }
 rewind(fp);
        Read degree of polynomial */
  fscanf(fp, "%d", &data_tmp_int);
  ADAT_BURN_SPEED_DEG = data_tmp_int;
  fgets(descript, 80, fp);
  P(printf("adat_miss_poly_deg(0) is%3d %s",
        ADAT_BURN_SPEED_DEG, descript););
        Read array data
                          */
  i=0;
  while(fscanf(fp, "%f", &data_tmp) != EOF){
        adat_burn_speed_coeff[i] = data_tmp;
        fgets(descript, 80, fp);
        P(printf("adat_burn_speed_coeff(%3d) is%11.3f %s", i,
                    adat_burn_speed_coeff[i], descript););
        ++i;
  }
  fclose(fp);
END DEFAULT BURN SPEED DATA FOR miss_adat.c READ FROM FILE
                                                                */
DEFAULT COAST SPEED DATA FOR miss_adat.c READ FROM FILE
  fp = fopen("/simnet/data/ms_ad_cs.d","r");
  if(fp==NULL){
        fprintf(stderr, "Cannot open /simnet/data/ms_ad_cs.d\n");
        exit();
  }
  rewind(fp);
        Read degree of polynomial */
  /*
  fscanf(fp, "%d", &data_tmp_int);
  ADAT_COAST_SPEED_DEG = data_tmp_int;
  fgets(descript, 80, fp);
  P(printf("adat_miss_poly_deg(1) is%3d %s",
        ADAT_COAST_SPEED_DEG, descript););
        Read array data
  i=0;
  while(fscanf(fp,"%f", &data_tmp) != EOF){
         adat_coast_speed_coeff[i] = data_tmp;
         fgets(descript, 80, fp);
         P(printf("adat_coast_speed_coeff(%3d) is%11.3f %s", i,
                     adat_coast_speed_coeff(i], descript););
         ++i;
   }
```

```
fclose(fp);
END DEFAULT COAST SPEED DATA FOR miss_adat.c READ FROM FILE
                                                                 */
                                                                 */
DEFAULT BURN TURN DATA FOR miss_adat.c READ FROM FILE
   fp = fopen("/simnet/data/ms_ad_bt.d", "r");
   if(fp==NULL){
         fprintf(stderr, "Cannot open /simnet/data/ms_ad_bt.d\n");
         exit();
   }
   rewind(fp);
         Read degree of polynomial */
   fscanf(fp, "%d", &data_tmp_int);
   ADAT_BURN_TURN_DEG = data_tmp_int;
   fgets(descript, 80, fp);
   P(printf("adat_miss_poly_deg(2) is%3d %s",
         ADAT_BURN_TURN_DEG, descript););
         Read array data
                          */
   i=0:
   while(fscanf(fp, "%f", &data_tmp) != EOF){
         adat_burn_turn_coeff[i] = data_tmp;
          fgets(descript, 80, fp);
         P(printf("adat_burn_turn_coeff(%3d) is%11.3f %s", i,
                      adat_burn_turn_coeff[i], descript););
          ++i;
    }
    fclose(fp);
END DEFAULT BURN TURN DATA FOR miss_adat.c READ FROM FILE
                                                                 */
                                                                 */
 DEFAULT COAST TURN DATA FOR miss_adat.c READ FROM FILE
    fp = fopen("/simnet/data/ms_ad_ct.d", "r");
    if(fp==NULL){
          fprintf(stderr, "Cannot open /simnet/data/ms_ad_ct.d\n");
          exit();
    }
    rewind(fp);
          Read degree of polynomial */
    fscanf(fp, "%d", &data_tmp_int);
    ADAT_COAST_TURN_DEG = data_tmp_int;
    fgets(descript, 80, fp);
    P(printf("adat_miss_poly_deg(3) is%3d %s",
          ADAT_COAST_TURN_DEG, descript););
          Read array data
    /*
    i=0:
```

```
while(fscanf(fp, "%f", &data_tmp) != EOF){
           adat_coast_turn_coeff[i] = data_tmp;
           fgets(descript, 80, fp);
           P(printf("adat_coast_turn_coeff(%3d) is%11.3f %s", i,
                       adat_coast_turn_coeff[i], descript););
     }
     fclose(fp);
  END DEFAULT COAST TURN DATA FOR miss_adat.c READ FROM FILE
                                                                  */
  DEFAULT TEMP BIAS DATA FOR miss_adat.c READ FROM FILE
                                                                  */
     fp = fopen("/simnet/data/ms_ad_tb.d","r");
     if(fp==NULL){
           fprintf(stderr, "Cannot open /simnet/data/ms_ad_tb.d\n");
           exit();
     }
     rewind(fp);
           Read degree of polynomial */
     fscanf(fp, "%d", &data_tmp_int);
     ADAT_TEMP_BIAS_DEG = data_tmp_int;
     fgets(descript, 80, fp);
     P(printf("adat_miss_poly_deg(4) is%3d %s",
           ADAT_TEMP_BIAS_DEG, descript););
           Read array data */
     i=0:
     while(fscanf(fp, "%f", &data_tmp) != EOF){
           adat_temp_bias_coeff[i] = data_tmp;
           fgets(descript, 80, fp);
           P(printf("adat_temp_bias_coeff(%3d) is%11.3f %s", i,
                        adat_temp_bias_coeff[i], descript););
           ++i;
     }
     fclose(fp);
/* END DEFAULT TEMP BIAS DATA FOR miss_adat.c READ FROM FILE
                                                                   */
   num_adats = num_missiles;
   adat_array = missile_array;
   for (i = 0; i < num_missiles; i++)</pre>
   {
        adat_array[i].mptr.state = ADAT_FREE;
       adat_array[i].mptr.max_flight_time = ADAT_MAX_FLIGHT_TIME;
        adat_array[i].mptr.max_turn_directions = 1;
   Initialize the proximity fuze.
   missile_fuze_prox_init ();
```

```
Initialize the tube to sight transformation matricies.
/*/
   mag = sqrt (adat_burn_speed_coeff[0] * adat_burn_speed_coeff[0] +
           2.0 * adat_temp_bias_coeff[0] * adat_temp_bias_coeff[0]);
   tube_C_sight_right[1][0] = adat_temp_bias_coeff[0] / mag;
   tube_C_sight_right[1][1] = adat_burn_speed_coeff[0] / mag;
    tube_C_sight_right[1][2] = adat_temp_bias_coeff[0] / mag;
   mag = sqrt (tube_C_sight_right[1][0] * tube_C_sight_right[1][0] +
           tube_C_sight_right[1][1] * tube_C_sight_right[1][1]);
    tube_C_sight_right[0][0] = tube_C_sight_right[1][1] / mag;
    tube_C_sight_right[0][1] = -tube_C_sight_right[1][0] / mag;
    tube_C_sight_right[0][2] = 0.0;
    tube_C_sight_right[2][0] = tube_C_sight_right[1][2] *
           tube_C_sight_right[0][1];
    tube_C_sight_right[2][1] = -tube_C_sight_right[1][2] *
           tube_C_sight_right[0][0];
    tube C_sight_right[2][2] = mag;
   mat_copy (tube_C_sight_right, tube_C_sight_left);
    tube_C_sight_left[0][1] = -tube_C_sight_left[0][1];
    tube C_sight_left[1][0] = -tube_C_sight_left[1][0];
    tube_C_sight_left[2][0] = -tube_C_sight_left[2][0];
} *
int missile_adat_is_free( missile )
int missile;
    return( (adat_array[missile].mptr.state == ADAT_FREE ));
           ***************
  ROUTINE: missile_adat_fire
                  aptr - A pointer to the ADAT missile to be
   PARAMETERS:
                   fired.
            target_type - The missile can be set for three *
                          types of targets by the launching *
                          vehicle. This variable stores
                          the setting.
            launch_point - The location in world
                           coordinates that the missile is
                           launched from.
            loc_sight_to_world - The sight to world
                                 transformation matrix used *
                                 only in this routine.
            launch_speed - The speed of the launch
                           platform (assumed to be in the
                           direction of the missile). *
            range_to_intercept - Range to intercept.
            tube - The tube the missile was launched from.
            target_vehicle_id - The vehicle ID of the *
                                target (if any).
 * RETURNS: TRUE if successful, FALSE if not.
 * PURPOSE: This routine performs the functions
            specifically related to the firing of a ADAT
```

```
missile.
int missile_adat_fire (aptr, target_type, launch_point, loc_sight_to_world,
        launch_speed, range_to_intercept, tube, target_vehicle_id)
ADAT_MISSILE *aptr;
int target_type;
VECTOR launch_point;
T_MATRIX loc_sight_to_world;
REAL launch_speed;
REAL range_to_intercept;
int tube;
VehicleID *target_vehicle_id;
{
                              /* A counter. */
    int i;
                              /* Pointer to the particular generic missile
    MISSILE *mptr;
                                pointed at by _aptr_. */
                              /* Indication of whether target is known. */
    int comm_target_type;
/*/
 * Find _mptr_ and _target_id_.
    mptr = &(aptr->mptr);
    if (target_vehicle_id == 0)
        aptr->target_vehicle_id.vehicle = vehicleIrrelevant;
        aptr->target_vehicle_id = *target_vehicle_id;
    Set the initial time, location, orientation, and speed of the generic
    missile.
 /*/
    mptr->time = 0.0;
     vec_copy (launch_point, mptr->location);
     if (range_to_intercept < CLOSE_RANGE)</pre>
        mat_copy (loc_sight_to_world, mptr->orientation);
     else
         if (((tube / 2) * 2) == tube)
             mat_mat_mul (tube_C_sight_left, loc_sight_to_world,
                     mptr->orientation);
         else
             mat_mat_mul(tube_C_sight_right, loc_sight_to_world,
                     mptr->orientation);
     mptr->speed = missile_util_eval_poly (ADAT_BURN_SPEED_DEG,
             adat_burn_speed_coeff, 0.0) + launch_speed;
     mptr->init_speed = launch_speed;
 /*/
  * Indicate that the proximity fuze has no vehicles it is tracking.
 /*/
     aptr->pptr = NULL;
 /*/
  * Set fuze distance and fuze target according to missile target
  * setting. Set network variables.
 /*/
```

```
switch (target_type)
   case ADAT_TGT_GND:
       aptr->fuze_dist_sq = 0.0;
       aptr->target_flag = PROX_FUZE_ON_NO_VEH;
       break;
   case ADAT_TGT_HELO:
       aptr->fuze_dist_sq = HELO_FUZE_DIST_SQ;
       if (aptr->target_vehicle_id.vehicle == vehicleIrrelevant)
           aptr->target_flag = PROX_FUZE_ON_ALL_VEH;
       else
           aptr->target_flag = PROX_FUZE_ON_ONE_VEH;
       break;
   case ADAT_TGT_AIR:
       aptr->fuze_dist_sq = AIR_FUZE_DIST_SQ;
       if (aptr->target_vehicle_id.vehicle == vehicleIrrelevant)
           aptr->target_flag = PROX_FUZE_ON_ALL_VEH;
       else
           aptr->target_flag = PROX_FUZE_ON_ONE_VEH;
       break;
   default:
       aptr->fuze_dist_sq = 0.0;
       aptr->target_flag = PROX_FUZE_ON_NO_VEH;
       printf ("MISS_ADAT: Unknown target type %d\n", target_type);
       break;
   if (aptr->target_vehicle_id.vehicle == vehicleIrrelevant)
       comm_target_type = targetUnknown;
   else
       comm_target_type = targetIsVehicle;
/*/
   Tell the rest of the world about the firing of the missile. If this
   cannot be done, return FALSE.
/*/
   if (!missile_util_comm_fire_missile (mptr, MSL_TYPE_MISSILE,
            map_get_ammo_entry_from_network_type (munition_US_ADATS),
            munition_US_ADATS, munition_US_ADATS, & (aptr->target_vehicle_id),
            comm_target_type, objectIrelevant, tube))
        return (FALSE);
/*/
   If all was successful, put any flying missiles in an unguided state
   and put this missile in a guided state.
/*/
    for (i = 0; i < num_adats; i++)</pre>
        if ((adat_array[i].mptr.state == ADAT_GUIDE) ||
                (adat_array[i].mptr.state === ADAT_CLOSE))
            adat_array[i].mptr.state = ADAT_UNGUIDE;
    if (range_to_intercept < CLOSE_RANGE)
        mptr->state = ADAT_CLOSE;
        mptr->state = ADAT_GUIDE;
    return (TRUE);
}
```

```
********
* ROUTINE: missile_adat_fly_missiles
* PARAMETERS: sight_location - The location in world
                            coordinates of the gunner's
                            sight.
           loc_sight_to_world - The sight to world
                               transformation matrix used *
                                only in this routine.
               veh_list - Vehicle list ID.
 * RETURNS: none
* PURPOSE: This routine flies out all missiles in a
           flying state.
void missile_adat_fly_missiles (sight_location, loc_sight_to_world, veh_list)
VECTOR sight_location;
T_MATRIX loc_sight_to_world;
int veh_list;
    int i; /* A counter. */
/*/
 * Fly out all flying missiles.
    for (i = 0; i < num_adats; i++)
        if (adat_array[i].mptr.state != ADAT_FREE)
           missile_adat_fly (&(adat_array[i]), sight_location,
                    loc_sight_to_world, i, veh_list);
    }
}
 * ROUTINE: missile_adat_fly
 * PARAMETERS: aptr - A pointer to the ADAT missile that is to *
                  be flown out.
            sight_location - The location in world
                             coordinates of the gunner's
                            sight.
            loc_sight_to_world - The sight to world
                                transformation matrix used *
                                only in this routine.
            tube - The tube the missile was launched from. *
                veh_list - Vehicle list ID.
 * RETURNS: none
  * PURPOSE: This routine performs the functions
            specifically related to the flying a ADAT \star
            missile.
```

```
void missile_adat_fly (aptr, sight_location, loc_sight_to_world, tube,
    veh_list)
ADAT_MISSILE *aptr;
VECTOR sight_location;
T_MATRIX loc_sight_to_world;
int tube;
int veh_list;
                          /* A pointer to the generic aspects of _aptr_. */
    MISSILE *mptr;
                          /* The current time after launch (ticks). */
    REAL time;
                          /* The value of the temporal bias. */
    REAL bias;
/*/
    Set _mptr_ and _time_. These values are created mostly for increased
    readablity.
    mptr = &(aptr->mptr);
    time = mptr->time;

    Find the current missile speed and the cosines of the maximum allowed turn

    angles in each direction. The equations used are different before and
   after motor burnout.
1*/
    if (time < ADAT_BURNOUT_TIME)
        mptr->speed = missile_util_eval_poly (ADAT_BURN_SPEED_DEG,
                 adat_burn_speed_coeff, time) + mptr->init_speed;
        mptr->cos_max_turn[0] = missile_util_eval_poly (ADAT_BURN_TURN_DEG,
                 adat_burn_turn_coeff, time);
     }
    else
     {
         mptr->speed = missile_util_eval_poly (ADAT_COAST_SPEED_DEG,
                 adat_coast_speed_coeff, time) + mptr->init_speed;
         mptr->cos_max_turn[0] = missile_util_eval_poly (ADAT_COAST_TURN_DEG,
                 adat_coast_turn_coeff, time);
     }
 /*/
  * Find the target point, etc.
     if ((mptr->state == ADAT_GUIDE) || (mptr->state == ADAT_CLOSE))
     {
         if ((time < ADAT_TEMP_BIAS_TIME) && (mptr->state == ADAT_GUIDE))
             bias = missile_util_eval_poly (ADAT_TEMP_BIAS_DEG,
                     adat_temp_bias_coeff, time);
             if (((tube / 2) * 2) == tube)
                 missile_target_los_bias (mptr, sight_location,
                          loc_sight_to_world, -bias, bias);
                 missile_target_los_bias (mptr, sight_location,
                          loc_sight_to_world, bias, bias);
         }
         else
             missile_target_los (mptr, sight_location, loc_sight_to_world);
     }
```

```
else if (mptr->state == ADAT_UNGUIDE)
       missile_target_unguided (mptr);
   else
       printf ("MISSILE_ADAT: disallowed missile state %d\n", mptr->state);
/*/
   Try to actually fly the missile. If this fails stop the missile altogether
   and return.
/*/
    if (!missile_util_flyout (mptr))
       missile_adat_stop (aptr);
        return;
    }
    else
    {
/*/
        If the missile successfully flew, process the proximity fuze.
/*/
        missile_fuze_prox (mptr, MSL_TYPE_MISSILE, aptr->target_flag,
                &(aptr->target_vehicle_id), &(aptr->pptr), veh_list,
                INVEST_DIST_SQ, aptr->fuze_dist_sq);
        If the missile successfully flew, check for an intersection with the
        ground or a vehicle. If one is found, blow up the missile, stop its
        flyout and return.
/*/
        if (missile_util_comm_check_detonate (mptr, MSL_TYPE_MISSILE))
            missile_adat_stop (aptr);
            return;
/*/
    If the missile is to continue to fly, return.
/*/
    return;
 }
  * ROUTINE: missile_adat_reset_missiles
                   none
  * PARAMETERS:
  * RETURNS: none
  * PURPOSE: This routine puts any flying missile into an
             unquided state.
 void missile_adat_reset_missiles ()
                 /* A counter. */
     int i;
 /*/
  * Reset all flying missiles.
 /*/
     for (i = 0; i < num_addts; i++)</pre>
```

```
{
       if ((adat_array[i].mptr.state == ADAT_GUIDE) ||
              (adat_array[i].mptr.state == ADAT_CLOSE))
          adat_array[i].mptr.state = ADAT_UNGUIDE;
   }
}
        **************
 * ROUTINE: missile_adat_stop
              aptr - A pointer to the ADAT missile that is to *
 * PARAMETERS:
                be stopped.
 * RETURNS: none
 * PURPOSE: This routine causes all concerned to forget
           about the missile. It should be called when
           the flyout of any ADAT missile is stopped *
           (whether or not it has exploded). Note that
           this routine can only be called within this
           module.
      ********
static void missile_adat_stop (aptr)
ADAT_MISSILE *aptr;
{
/*/
    Tell the world to stop worrying about this missile then release the
    memory for use by other missiles.
    missile_fuze_prox_stop (&(aptr->pptr));
    missile_util_comm_stop_missile (&(aptr->mptr), MSL_TYPE_MISSILE);
    aptr->mptr.state = ADAT_FREE;
}
```

# Appendix F - Source code listing for miss\_atgm.c.

The following appendix contains the source code listing for miss\_atgm.c for convenience in document maintenance and understanding of the CSU.

```
/* $Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/libmissile/RCS/miss_atg
m.c, v 1.4 1993/01/28 23:22:08 cm-adst Exp $ */
 * $Log: miss_atgm.c,v $
 * Revision 1.4 1993/01/28 23:22:08 cm-adst
 * P.DesMeules changes for spcr 31
 * Revision 1.3 1993/01/06 21:12:37 cm-adst
 * R.Branson's changes for the weapons model.
 * Revision 1.1 1992/09/30 16:39:52 cm-adst
 * Initial Version
*/
static char RCS_ID[] = "$Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/li
bmissile/RCS/miss_atgm.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $";
* Revisions:
                                                               SP/CR Number
                                       Title
                  Date
                             Author
      Version
                   10/23/92 R. Branson Data File Initiali-
     1.2
                                           zation
                  10/30/92 R. Branson Added pathname to data
     1.3
                                           directory
                   11/25/92 R. Branson Changed %i to %d
     1.4
                   01/19/93 P.Desmeules Increased the size of the
                                                                     31
      1.5
                                           fgets to make sure the
                                         ..whole line is read in.
                        Description of Modification
        SP/CR No.
                        Hard coded defines changed to array elements.
                        Characteristics/parameter data array added.
                        Degree of polynomial data array added.
                        Added file reads for ATGM characteristics/
                           parameters, burn speed coefficients, coast speed
                           coefficients, burn turn coefficients, and coast
                           turn coefficients.
                        Added "/simnet/data/" to each data file pathname.
            *****************
   * FILE: miss_atgm.c
```

```
Bryant Collard
 * AUTHOR:
 * MAINTAINER: Bryant Collard
              This missile is the same as the tow except
 * PURPOSE:
                 it uses point targeting. It flys to a point
                 rather than the view direction
             10/31/88 bryant: Creation
 * HISTORY:
               4/26/89 bryant: Added statically allocated mem
 * Copyright (c) 1988 BBN Systems and Technologies, Inc.
 * All rights reserved.
 ******************************
 #include "stdio.h"
 #include "sim_types.h"
 #include "sim_dfns.h"
 #include "basic.h"
 #include "mun_type.h"
 #include "libmatrix.h"
#include "libmap.h"
 #include "librva.h"
 #include "miss_atgm.h"
 #include "libmiss_dfn.h"
 #include "libmiss_loc.h"
 * Debug macro
 #ifdef FILEDBG
 #define P(a)
 #define P(a)
 #endif
  * Define missile characteristics.
 /*/
 #define TOW_RANGE_LIMIT_TIME tow_miss_char[1]
 #define TOW_MAX_FLIGHT_TIME tow_miss_char[2]
 #define ATGM_TURN_FACTOR tow_miss_char[3]
 /*/

    The following terms set the order of the polynomials used to determine

  * the speed or cosine of the maximum allowed turn rate of the missile
  * at any point in time.
 1*/
  #define TOW_BURN_SPEED_DEG tow_miss_poly_deg[0]
```

```
#define TOW_COAST_SPEED_DEG tow_miss_poly_deg[1]
#define TOW_BURN_TURN_DEG tow_miss_poly_deg[2]
#define TOW_COAST_TURN_DEG tow_miss_poly_deg[3]
/*/
* Tow missile characteristic parameters initialized to default values.
/*/
static REAL tow_miss_char[5] =
            /* ticks (1.6 sec) */
   24.0,
          /* ticks (17.89 sec) */
  268.35,
          /* ticks - cos of max turn > 1.0 beyond this point */
  200.00,
           /* ATGM turn factor for wider turning capability */
    0.9,
    0.0
};
/*/
 * The following terms set the order of the polynomials used to determine
 * the speed and turn of the missile at any point in time.
/*/
static int tow_miss_poly_deg[5] =
            /* Speed before motor burnout. */
   2,
            /* Speed after motor burnout. */
   3,
            /* Cosine of max turn before burnout. */
   1,
            /* Cosine of max turn after burnout. */
   3,
            /* not used. */
   0
};
 * Coefficients for the speed polynomial before motor burnout initialized to
 * default values.
/*/
static REAL tow_burn_speed_coeff[5] =
{
                         /* a_0 - m/tick (67.0 m/sec) */
/* a_1 - m/tick**2 (274.9732662 m/sec**2) */
    4.466666667,
    1.222103405,
                         /* a_2 - m/tick**3 (-82.7057910 m/sec**3) */
   -0.024532086,
    0.0,
    0.0
};
 * Coefficients for the speed polynomial after motor burnout initialized to
 * default values.
 /*/
 static REAL tow_coast_speed_coeff[5] =
                         /* a_0 - m/tick (327.2858074 m/sec) */
/* a_1 - m/tick**2 (-21.4609544 m/sec**2) */
     21.81905383,
     -9.5382019e-2,
                         /* a_2 - m/tick**3 ( 0.8227650 m/sec**3) */
     2.4378222e-4,
                          /* a_3 - m/tick**4 ( -0.0133200 m/sec**4) */
     -2.6311111e-7,
      0.0
```

```
};
/*/
* Coefficients for the cosine of max turn polynomials before motor burnout.
* The structure _MAX_COS_COEFF_ is used to store the values for the turn
* sideways, up, and down polynomials along with their order.
static MAX_COS_COEFF tow_burn_turn_coeff =
{
                         /* Order of the polynomials. */
    1,
    {
                         /* Sidewards turn. */
                        /* a_0 - cos(rad)/tick */
        0.999976868652,
       -3.5933955e-7
                        /* a_1 - cos(rad)/tick**2 */
    },
                         /* Upwards turn. */
        0.999960667258, /* a_0 - cos(rad)/tick */
                        /* a_1 - cos(rad)/tick**2 */
       -3.1492328e-6
    },
    {
                         /* Downwards turn. */
        0.999978909989, /* a_0 - cos(rad)/tick */
                        /* a_1 - cos(rad)/tick**2 */
       -7.8194991e-9
    }
};
 * Coefficients for the cosine of max turn polynomials after motor burnout.
/*/
static MAX_COS_COEFF tow_coast_turn_coeff =
                         /* Order of the polynomials. */
    3,
    {
                         /* Sidewards turn. */
                         /* a_0 - cos(rad)/tick */
        0.99995112518,
                         /* a_1 - cos(rad)/tick**2 */
        8.96333e-7,
                        /* a_2 - cos(rad)/tick**3 */
       -5.995375e-9,
                         /* a_3 - cos(rad)/tick**4 */
        1.162225e-11
    },
                         /* Upwards turn. */
                         /* a_0 - cos(rad)/tick */
        0.9998498495,
                        /* a_1 - cos(rad)/tick**2 */
        1.657779e-6,
                        /* a_2 - cos(rad)/tick**3 */
       -8.231861e-9,
                        /* a_3 - cos(rad)/tick**4 */
        1.381832e-11
    },
                         /* Downwards turn. */
                         /* a_0 - cos(rad)/tick */
        0.9999714014,
                        /* a_1 - cos(rad)/tick**2 */
        3.382077e-7,
                        /* a_2 - cos(rad)/tick**3 */
       -1.601259e-9,
                        /* a_3 - cos(rad)/tick**4 */
        2.623014e-12
```

```
}
};
* Declare static functions.
/*/
static void missile_atgm_stop ();
 * ROUTINE: missile_atgm_init
 * PARAMETERS: tptr - a pointer to the TOW to be
                  initialized.
 * RETURNS: none
 * PURPOSE: This routine initializes the state of the
           missile to indicate that it is available and
           sets values that never change.
 *************
void missile_atgm_init (tptr)
ATGM_MISSILE *tptr;
{
      int
            i;
          data_tmp_int;
      int
      float data_tmp;
      char descript[80];
      FILE *fp;
      P(printf("$$$$ ATGM missile file data $$$$\n"););
                                                                       */
/* DEFAULT CHARACTERISTICS DATA FOR miss_atgm.c READ FROM FILE
      fp = fopen("/simnet/data/ms_at_ch.d", "r");
       if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_at_ch.d\n");
            exit();
       }
       rewind(fp);
            Read array data */
       i=0;
       while(fscanf(fp, "%f", &data_tmp) != EOF){
             tow_miss_char[i] = data_tmp;
             fgets(descript, 80, fp);
             P(printf("tow_miss_char(%3d) is%11.3f %s", i, tow_miss_char[i],
                        descript););
             ++i;
       fclose(fp);
 /* END DEFAULT CHARACTERISTICS DATA FOR miss_atgm.c READ FROM FILE
                                                                        */
```

. ;

```
/* DEFAULT BURN SPEED DATA FOR miss_atgm.c READ FROM FILE
                                                                   */
      fp = fopen("/simnet/data/ms_at_bs.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_at_bs.d\n");
            exit();
      }
      rewind(fp);
            Read degree of polynomial */
      fscanf(fp, "%d", &data_tmp_int);
      TOW_BURN_SPEED_DEG = data_tmp_int;
      fgets(descript, 80, fp);
      P(printf("tow_miss_poly_deg(0) is%3d %s", TOW_BURN_SPEED_DEG,
                descript););
            Read array data
      i=0:
      while(fscanf(fp, "%f", &data_tmp) != EOF){
            tow_burn_speed_coeff[i] = data_tmp;
             fgets(descript, 80, fp);
            P(printf("tow_burn_speed_coeff(%3d) is%11.3f %s", i,
                        tow_burn_speed_coeff[i], descript););
             ++i;
      }
       fclose(fp);
    END DEFAULT BURN SPEED DATA FOR miss_atgm.c READ FROM FILE
/* DEFAULT COAST SPEED DATA FOR miss_atgm.c READ FROM FILE
                                                                    */
       fp = fopen("/simnet/data/ms_at_cs.d","r");
       if(fp==NULL){
             fprintf(stderr, "Cannot open /simnet/data/ms_at_cs.d\n");
             exit();
       }
       rewind(fp);
             Read degree of polynomial */
       fscanf(fp, "%d", &data_tmp_int);
       TOW_COAST_SPEED_DEG = data_tmp_int;
       fgets(descript, 80, fp);
       P(printf("tow_miss_poly_deg(1) is%3d %s", TOW_COAST_SPEED_DEG,
                 descript););
       /*
             Read array data
       i=0;
       while(fscanf(fp, "%f", &data_tmp) != EOF){
             tow_coast_speed_coeff[i] = data_tmp;
             fgets(descript, 80, fp);
```

```
P(printf("tow_coast_speed_coeff(%3d) is%11.3f %s", i,
                    tow_coast_speed_coeff[i], descript););
        ++i;
  }
  fclose(fp);
END DEFAULT COAST SPEED DATA FOR miss_atgm.c READ FROM FILE
                                                                * /
DEFAULT BURN TURN DATA FOR miss_atgm.c READ FROM FILE
                                                                */
  fp = fopen("/simnet/data/ms_at_bt.d","r");
  if(fp==NULL){
        fprintf(stderr, "Cannot open /simnet/data/ms_at_bt.d\n");
        exit();
  }
  rewind(fp);
        Read degree of polynomial */
  fscanf(fp, "%d", &data_tmp_int);
  TOW_BURN_TURN_DEG = data_tmp_int;
  tow_burn_turn_coeff.deg = data_tmp_int;
  fgets(descript, 80, fp);
  P(printf("tow_miss_poly_deg(2) is%3d %s", TOW_BURN_TURN_DEG,
            descript););
        Read array data */
  for (i=0; i <= data_tmp_int; i++) {</pre>
         fscanf(fp, "%f", &data_tmp);
         tow_burn_turn_coeff.side_coeff[i] = data_tmp;
         fgets(descript, 80, fp);
        P(printf("tow_burn_turn_coeff.side_coeff(%3d) is%11.3f %s", i,
                     tow_burn_turn_coeff.side_coeff[i], descript););
   }
   for (i=0; i <= data_tmp_int; i++) {
         fscanf(fp, "%f", &data_tmp);
         tow_burn_turn_coeff.up_coeff[i] = data_tmp;
         fgets(descript, 80, fp);
         P(printf("tow_burn_turn_coeff.up_coeff(%3d) is%11.3f %s", i,
                     tow_burn_turn_coeff.up_coeff[i], descript););
   }
   for (i=0; i <= data_tmp_int; i++) {</pre>
         fscanf(fp, "%f", &data_tmp);
         tow_burn_turn_coeff.down_coeff[i] = data_tmp;
         fgets(descript, 80, fp);
         P(printf("tow_burn_turn_coeff.down_coeff(%3d) is%11.3f %s", i,
                     tow_burn_turn_coeff.down_coeff[i], descript););
   }
   fclose(fp);
END DEFAULT BURN TURN DATA FOR miss_atgm.c READ FROM FILE
                                                                 */
```

```
*/
/* DEFAULT COAST TURN DATA FOR miss_atgm.c READ FROM FILE
     fp = fopen("/simnet/data/ms_at_ct.d", "r");
     if(fp==NULL){
           fprintf(stderr, "Cannot open /simnet/data/ms_at_ct.d\n");
     rewind(fp);
           Read degree of polynomial */
      fscanf(fp, "%d", &data_tmp_int);
     TOW_COAST_TURN_DEG = data_tmp_int;
      tow_coast_turn_coeff.deg = data_tmp_int;
      fgets(descript, 80, fp);
      P(printf("tow_miss_poly_deg(3) is%3d %s", TOW_COAST_TURN_DEG,
                descript););
         Read array data
      for (i=0; i <= data_tmp_int; i++) {</pre>
            fscanf(fp, "%f", &data_tmp);
            tow_coast_turn_coeff.side_coeff[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("tow_coast_turn_coeff.side_coeff(%3d) is%11.3f %s", i,
                        tow_coast_turn_coeff.side_coeff[i], descript););
      }
      for (i=0; i <= data_tmp_int; i++) {</pre>
            fscanf(fp, "%f", &data_tmp);
            tow_coast_turn_coeff.up_coeff[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("tow_coast_turn_coeff.up_coeff(%3d) is%11.3f %s", i,
                        tow_coast_turn_coeff.up_coeff[i], descript););
      }
      for (i=0; i <= data_tmp_int; i++) {
            fscanf(fp, "%f", &data_tmp);
            tow_coast_turn_coeff.down_coeff[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("tow_coast_turn_coeff.down_coeff(%3d) is%11.3f %s", i,
                        tow_coast_turn_coeff.down_coeff[i], descript););
      }
      fclose(fp);
   END DEFAULT COAST TURN DATA FOR miss_atgm.c READ FROM FILE
    tptr->mptr.state = FALSE;
    tptr->mptr.max_flight_time = TOW_MAX_FLIGHT_TIME;
    tptr->mptr.max_turn_directions = 3;
    /********************
    /* change turn polynomial coefficients so missile has larger */
    /* max turn angle. Sirce Ph determines when a vehicle should be
                                                                         */
    /* impacted, turn rates should not effect missile effectiveness
```

```
for (i=0; i<tow_burn_turn_coeff.deg; i++)</pre>
     tow_burn_turn_coeff.side_coeff[i] *= ATGM_TURN_FACTOR;
     tow_burn_turn_coeff.up_coeff[i] *= ATGM_TURN_FACTOR;
     tow_burn_turn_coeff.down_coeff[i] *= ATGM_TURN_FACTOR;
   for (i=0; i<tow_coast_turn_coeff.deg; i++)</pre>
     tow_coast_turn_coeff.side_coeff[i] *= ATGM_TURN_FACTOR;
     tow_coast_turn_coeff.up_coeff[i] *= ATGM_TURN_FACTOR;
     tow_coast_turn_coeff.down_coeff[i] *= ATGM_TURN_FACTOR;
}
      *************
 * ROUTINE: missile_atgm_fire
 * PARAMETERS: tptr - A pointer to the TOW missile to be *
                 fired.
                launch_point - The location in world
 * PARAMETERS:
                         coordinates that the missile is *
                         launched from.
           loc_sight_to_world - The sight to world
                              transformation matrix used *
                              only in this routine.
           launch_speed - The speed of the launch
                         platform (assumed to be in the
                         direction of the missile). *
           tube - The tube the missile was launched from.
 * RETURNS: none
 * PURPOSE: This routine performs the functions
           specifically related to the firing of a TOW
           missile.
ATGM_MISSILE *missile_atgm_fire (tptr, launch_point, loc_sight_to_world,
       launch_speed, tube, try_to_hit_target, target_id, target_loc)
ATGM_MISSILE *tptr;
VECTOR launch_point;
T_MATRIX loc_sight_to_world;
REAL launch_speed;
int tube;
int try_to_hit_target;
VehicleID target_id;
VECTOR target_loc;
    MISSILE *mptr; /* Pointer to the particular generic missile
                           pointed at by _tptr_. */
 * Find _mptr_.
    mptr = &(tptr->mptr);
```

```
/*/
* Set the initial time, location, orientation, and speed of the generic
 * missile.
/*/
   mptr->time = 0.0;
   vec_copy (launch_point, mptr->location);
   mat_copy (loc_sight_to_world, mptr->orientation);
   mptr->speed = missile_util_eval_poly (TOW_BURN_SPEED_DEG,
           tow_burn_speed_coeff, 0.0) + launch_speed;
   mptr->init_speed = launch_speed;
/*/
   Set the wire as uncut.
/*/
    tptr->wire_is_cut = FALSE;
/*/
    if we are trying to hit a target then save the target_id. Otherwise,
   save the target location (some point in space)
    tptr->try_to_hit_target = try_to_hit_target;
    if (try_to_hit_target)
      tptr->target_id = target_id;
    else
      vec_copy(target_loc, tptr->target_location);
      }
    Tell the rest of the world about the firing of the missile. If this
    cannot be done, return.
/*/
    if (!missile_util_comm_fire_missile (mptr, MSL_TYPE_MISSILE,
            map_get_ammo_entry_from_network_type (munition_US_TOW),
            munition_US_TOW, munition_US_TOW, NULL, targetUnknown,
            objectIrrelevant, tube))
        return;
 /*/
 * If all was successful, set the missile state to TRUE and return.
 /*/
    mptr->state = TRUE;
    return;
 }
        *****************
  * ROUTINE: missile_atgm_fly
                  tptr - A pointer to the TOW missile that is to *
  * PARAMETERS:
                   be flown out.
            sight_location - The location in world
                             coordinates of the gunner's
                             sight.
             loc_sight_to_world - The sight to world
                                 transformation matrix used *
                                 only in this routine.
  * RETURNS: none
```

```
* PURPOSE: This routine performs the functions
            specifically related to the flying a TOW
           missile.
void missile_atgm_fly (tptr, sight_location, loc_sight_to_world)
ATGM_MISSILE *tptr;
VECTOR sight_location;
T_MATRIX loc_sight_to_world;
                          /* A pointer to the generic aspects of _tptr_. */
    MISSILE *mptr;
                          /* The current time after launch (ticks). */
    REAL time;
    VehicleAppearanceVariant *target_vehicle;
                    /* pointer to target vehicles appearance packet */
    VECTOR target_plus_offset; /* this vector gives a targets location
                                   an appropriate offset for ground
                           with
                                    */
                           vehs
    static VECTOR ground_veh_offset = {0.0, 0.0, 1.0};
                        /* offset to aim missile at for ground vehs */
111
 * Set _mptr_ and _time_. These values are created mostly for increased
 * readablity.
/*/
    mptr = &(tptr->mptr);
    time = mptr->time;
/*/
 * If the missile has reached its maximum range (not the maximum distance
 * its allowed to fly), cut the wire.
/*/
    if ((time > TOW_RANGE_LIMIT_TIME) && !tptr->wire_is_cut)
        tptr->wire_is_cut = TRUE;
  Find the current missile speed and the cosines of the maximum allowed turn
 * angles in each direction. The equations used are different before and
 * after motor burnout.
    if (time < TOW_BURNOUT_TIME)</pre>
        mptr->speed = missile_util_eval_poly (TOW_BURN_SPEED_DEG,
                tow_burn_speed_coeff, time) + mptr->init_speed;
        missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
    }
    else
        mptr->speed = missile_util_eval_poly (TOW_COAST_SPEED_DEG,
                tow_coast_speed_coeff, time) + mptr->init_speed;
        missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
    }
/*/
 * If the wire has been cut, set the ground as the target; otherwise,
 * find a target point which will fly the missile along the gunner's line of
 * sight. This targeting scheme takes into account the errors introduced by
    attempting to guide the missile in a canted position.
```

```
/*/
   if (tptr->wire_is_cut)
     {
     printf("G");
       missile_target_ground (mptr);
     }
   else
     {
            if operator has successfully designated a target then
      * try_to_hit_target will be true. Therefore, we search the
      * list of targets for the vehicleID and fly missile to that
      * location.
            if try_to_hit_target is false then target point is passed
      * and we should fly the missile to the target_point.
            if try_to_hit_target is true and we can't find the
      * vehicle id in the rva list then the vehicle has dropped off the
       * net and we fly the missile into the ground.
      if (tptr->try_to_hit_target)
         if ((target_vehicle = rva_get_veh_app_pkt (&(tptr->target_id))) !=
           NULL)
           /st if the target is a ground vehicle we need to guide st/
           /* the missile to a point other than the center of mass
           /* for SIMNET ground vehicles the center of mass is on
           /* the ground. This causes missiles to fly into the */
           /* ground
           if ((target_vehicle->guises.distinguished &
               (objectDomainMask | vehicleEnvironmentMask)) ==
               (objectDomainVehicle | vehicleEnvironmentGround))
               vec_add (target_vehicle->location, ground_veh_offset,
                 target_plus_offset);
            else
               vec_copy (target_vehicle->location, target_plus_offset);
            missile_target_point(mptr, target_plus_offset);
            }
          else
            /* printf("g"); */
            missile_target_unguided (mptr);
          }
      else '
          /* printf("p"); */
                                 *********
```

```
/* guide the missile toward a point for 5 ticks, then just
        /* fly it straight ahead. With the wide turning radius */
        /* missile will fly around in circles otherwise
        /*************
        if (time < 5.0)
            missile_target_point(mptr, tptr->target_location);
        else
            missile_target_unguided (mptr);
     }
/*/
   Try to actually fly the missile. If this fails stop the missile altogether
* and return.
/*/
   if (!missile_util_flyout (mptr))
       missile_atgm_stop (tptr);
       return;
   }
   else
       If the missile successfully flew, check for an intersection with the
       ground or a vehicle. If one is found, blow up the missile, stop its
       flyout and return.
/*/
       if (missile_util_comm_check_intersection (mptr, MSL_TYPE_MISSILE))
           missile_util_comm_check_detonate (mptr, MSL_TYPE_MISSILE);
           missile_atgm_stop (tptr);
           return;
       }
    }
/*/
   If the missile is to continue to fly, return.
/*/
    return;
}
     *************
  ROUTINE: missile_atgm_stop
                 tptr - A pointer to the TOW missile that is to *
 * PARAMETERS:
                  be stopped.
 * RETURNS: none
 * PURPOSE: This routine causes all concerned to forget
           about the missile. It should be called when
           the flyout of any TOW missile is stopped *
            (whether or not it has exploded). Note that
           this routine can only be called within this
           module.
 *******************************
static void missile_atgm_stop (tptr)
```

```
ATGM_MISSILE *tptr;
{
/*/
* Tell the world to stop worrying about this missile then release the
* memory for use by other missiles.
/*/
   missile_util_comm_stop_missile (&(tptr->mptr), MSL_TYPE_MISSILE);
   tptr->mptr.state = FALSE;
}
/************************
 * ROUTINE: missile_atgm_cut_wire
 * PARAMETERS: tptr - A pointer to the TOW missile whose wire *
         is to be cut.
 * RETURNS: none
 * PURPOSE: This routine sets a flag indicating that the
          guidance wire of this missile is cut.
 void missile_atgm_cut_wire (tptr)
ATGM_MISSILE *tptr;
{
 * If the the wire is not already cut, cut the wire.
/*/
    if (!tptr->wire_is_cut)
       tptr->wire_is_cut = TRUE;
}
```

# Appendix G - Source code listing for miss\_hellfr.c.

The following appendix contains the source code listing for miss\_atgm.c for convenience in document maintenance and understanding of the CSU.

```
/* SHeader: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/libmissile/RCS/miss_hel
lfr.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $ */
 * $Log: miss_hellfr.c,v $
* Revision 1.4 1993/01/28 23:22:08 cm-adst
 * P.DesMeules changes for spcr 31
 * Revision 1.3 1993/01/06 00:45:01 cm-adst
 * R. Branson's weapon changes.
 * Revision 1.1 1992/09/30 16:39:52 cm-adst
 * Initial Version
static char RCS_ID[] = "$Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/li
bmissile/RCS/miss_hellfr.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $";
  ****************
  Revisions:
                                                         SP/CR Number
                                    Title
                          Author
                 Date
     Version
                 10/23/92 R. Branson Data File Initiali-
     1.2
                                      zation
                 10/30/92 R. Branson Added pathname to data
     1.3
                                      directory
                 11/25/92 R. Branson Changed %i to %d
     1.4
                  01/19/93 P.Desmeules Increased the size of the
      1.5
                                      fgets to make sure the
                                      whole line is read in.
     *****************
                    Description of Modification
       SP/CR No.
                     Hard coded defines changed to array elements.
                     Characteristics/parameter data array added.
                      Degree of polynomial data array added.
                      Added file reads for hellfire characteristics/
                        parameters, burn speed coefficients, coast speed
                        coefficients, and time-of-flight coefficients.
                      Added "/simnet/data/" to each data file pathname.
         **************************************
 /*****************************
             miss_hellfr.c
  * FILE:
             Bryant Collard
  * AUTHOR:
```

```
* MAINTAINER: Bryant Collard
             This file contains routines which fly out a
* PURPOSE:
              missile with the characteristics of a HELLFIRE
              missile.
              11/25/88 bryant: Creation
* HISTORY:
              4/24/89 bryant: Added static memory allocation
               08/07/90 bryant: NIU librva modifications.
              08/09/90 kris: corrected flight coefficients
* Copyright (c) 1988 BBN Systems and Technologies, Inc.
* All rights reserved.
*************************************
#include "stdio.h"
#include "math.h"
#include "sim_types.h"
#include "sim_dfns.h"
#include "basic.h"
#include "mun_type.h"
#include "libmatrix.h"
#include "libmap.h"
/*-- need Range_Squared info --*/
#include "libhull.h"
#include "libkin.h"
/*----*/
#include "miss_hellfr.h"
#include "libmissile.h"
#include "libmiss_dfn.h"
#include "libmiss_loc.h"
* Debug macro
#ifdef FILEDBG
#define P(a)
#else
#define P(a)
#endif
 * Define missile characteristics.
                                 hellfr_miss_char[ 0]
#define HELLFIRE_ARM_TIME
                                 hellfr_miss_char[ 1]
#define HELLFIRE_BURNOUT_TIME
#define HELLFIRE_MAX_FLIGHT_TIME hellfr_miss_char[ 2]
                                 hellfr_miss_char[ 3]
#define SPEED_0
                                 hellfr_miss_char[ 4]
#define THETA_0
/*/
 * Set parameters which will control flight trajectory behavior.
/*/
```

```
hellfr_miss_char[ 5]
#define SIN_UNGUIDE
                                  hellfr_miss_char[ 6]
#define COS_UNGUIDE
                                  hellfr_miss_char[ 7]
#define SIN_CLIMB
                                  hellfr_miss_char[ 8]
#define COS_CLIMB
                                  hellfr_miss_char[ 9]
#define SIN_LOCK
                                  hellfr_miss_char[10]
#define COS_LOCK
                                  hellfr_miss_char[11]
#define COS_TERM
                                  hellfr_miss_char[12]
#define COS_LOSE
/*/
   The following terms set the order of the polynomials used to determine
  the speed or cosine of the maximum allowed turn rate of the missile
 * at any point in time.
/*/
                                  hellfr_miss_poly_deg[ 0]
#define HELLFIRE_TOF_DEG
                                  hellfr_miss_poly_deg[ 1]
#define HELLFIRE_BURN_SPEED_DEG
#define HELLFIRE_COAST_SPEED_DEG hellfr_miss_poly_deg[ 2]
* Hellfire missile characteristic parameters initialized to default values.
/*/
static REAL hellfr_miss_char[15] =
                       /* ticks (1.3 sec) */
   20.0,
                       /* ticks (2.4 sec) */
   36.0,
                       /* ticks (36 sec)
  540.0,
                                           */
   30.95953043,
                       /* max_speed
    0.046542113,
                                           */
                       /* sin 4.0 deg
    0.069756474,
                                           */
                       /* cos 4.0 deg
    0.997564050,
                                           */
                       /* sin 3.5 deg
    0.004072424,
                                           */
                       /* cos 3.5 deg
    0.999991708,
                                           */
                       /* sin 9.0 deg
    0.156434465,
                       /* cos 9.0 deg
                                           */
    0.987688341,
                                           */
                       /* cos 76.0 deg
    0.241921896,
                       /* cos 20.0 deg
    0.939692621,
    0.0,
    0.0
};
 * Hellfire missile polynomial degree initialized to default values.
/*/
static int hellfr_miss_poly_deg[ 3] =
                                     */
        /* tof poly degree
    4,
         /* burn speed poly degree */
    3,
         /* coast speed poly degree */
};
 * Coefficients for the TOF polynomial initialized to default values.
static REAL hellfire_tof_coeff[10] =
```

```
*/ /* 1.2 seconds */
                      /* a_0 tick
  18.0,
                     /* a_1 tick/meter */
   3.1461816e-2,
                     /* a_2 tick/meter^2 */
   3.1921274e-6,
                      /* a_3 tick/meter^3 */
   3.5260413e-10,
                      /* a_4 tick/meter^4 */
  -2.8469594e-14,
                      /* a_5 tick/meter^5 */
   0.0,
                      /* a_6 tick/meter^6 */
   0.0,
                      /* a_7 tick/meter^7 */
   0.0,
                      /* a_8 tick/meter^8 */
   0.0,
                      /* a_9 tick/meter^9 */
   0.0
};
/*/
 * Coefficients for the speed polynomial before motor burnout initialized to
 * default values.
/*/
static REAL hellfire_burn_speed_coeff[10] =
                       /* a_0 - meters
     2.0044395e-2,
                                         */
                       /* a_1 - m/tick
     6.7384206e-1,
                       /* a_2 - m/tick^2 */
    9.8007701e-3,
                       /* a_3 - m/tick^3 */
   -1.6782227e-4,
                       /* a_4 - m/tick^4 */
     0.0.
                       /* a_5 - m/tick^5 */
     0.0,
                       /* a_6 - m/tick^6 */
     0.0,
                       /* a_7 - m/tick^7 */
     0.0,
                       /* a_8 - m/tick^8 */
     0.0,
                       /* a_9 - m/tick^9 */
     0.0
};
/*/
 * Coefficients for the speed polynomial after motor burnout initialized to
 * default values.
static REAL hellfire_coast_speed_coeff[10] =
{
                      /* a_0 - meters
     4.2738447e+1,
                       /* a_1 - m/tick */
    -4.1048613e-1,
                        /* a_2 - m/tick^2 */
    2.6023604e-3,
                        /* a_3 - m/tick^3 */
    -8.4870417e-6,
                        /* a_4 - m/tick^4 */
     1.3322932e-8,
                        /* a_5 - m/tick^5 */
    -7.9542005e-12,
                        /* a_6 - m/tick^6 */
     0.0,
                        /* a_7 - m/tick^7 */
     0.0,
                        /* a_8 - m/tick^8 */
     0.0,
                        /* a_9 - m/tick^9 */
     0.0
};
static ObjectType hellfire_ammo_type = munition_US_Hellfire;
static REAL
                                                                        */
    max_range_limit, /* [ MISSILE_US_MAX_RANGE_LIMIT ]
    max_range_squared, /* [ MISSILE_US_MAX_RANGE_LIMIT ^ 2 ]
                                                                        */
                                                                        */
                       /* [ MISSILE_US_SPEED_FACTOR ]
    speed_factor;
 /*/
```

```
* Declare static functions.
/*/
static void missile_hellfire_stop ();
/******************************
 * ROUTINE: missile_hellfire_init
                 mptr - a pointer to the HELLFIRE to be
 * PARAMETERS:
                  initialized.
 * RETURNS: none
 * PURPOSE: This routine initializes the state of the
           missile to indicate that it is available and
           sets values that never change.
 *************************************
void missile_hellfire_init (mptr)
MISSILE *mptr;
{
      int
           i;
           data_tmp_int;
      int
      float data_tmp;
      char descript[80];
      FILE *fp;
      P(printf("$$$$ HELLFIRE missile file data $$$$\n"););
/* DEFAULT CHARACTERISTIC DATA FOR miss_hellfr.c READ FROM FILE */
      fp = fopen("/simnet/data/ms_hf_ch.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_hf_ch.d\n");
            exit();
      }
      rewind(fp);
            Read array data */
      i=0;
      while(fscanf(fp, "%f", &data_tmp) != EOF)
            hellfr_miss_char[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("hellfr_miss_char(%3d) is%11.3f %s", i,
                       hellfr_miss_char[i], descript););
            ++i;
       }
       fclose(fp);
 /* END DEFAULT CHARACTERISTIC DATA FOR miss_hellfr.c READ FROM FILE
 /* DEFAULT TIME-OF-FLIGHT DATA FOR miss_hellfr.c READ FROM FILE */
       fp = fopen("/simnet/data/ms_hf_tf.d","r");
       if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_hf_tf.d\n");
```

```
exit();
     }
     rewind(fp);
           Read degree of polynomial
                                         */
       fscanf(fp, "%d", &data_tmp_int);
     hellfr_miss_poly_deg[0] = data_tmp_int;
     fgets(descript, 80, fp);
     P(printf("hellfr_miss_poly_deg(0) is%3d %s",
                       hellfr_miss_poly_deg[0], descript););
           Read array data */
     /*
     i=0;
     while(fscanf(fp,"%f", &data_tmp) != EOF)
           hellfire_tof_coeff[i] = data_tmp;
           fgets(descript, 80, fp);
           P(printf("hellfire_tof_coeff(%3d) is%11.3f %s", i,
                       hellfire_tof_coeff[i], descript););
     }
      fclose(fp);
/* END DEFAULT TIME-OF-FLIGHT DATA FOR miss_hellfr.c READ FROM FILE
/* DEFAULT BURN SPEED DATA FOR miss_hellfr.c READ FROM FILE */
      fp = fopen("/simnet/data/ms_hf_bs.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_hf_bs.d\n");
            exit();
      rewind(fp);
            Read degree of polynomial
        fscanf(fp, "%d", &data_tmp_int);
      hellfr_miss_poly_deg[1] = data_tmp_int;
      fgets(descript, 80, fp);
      P(printf("hellfr_miss_poly_deg(1) is%3d %s",
                        hellfr_miss_poly_deg[1], descript););
            Read array data
      /*
      i=0;
      while(fscanf(fp, "%f", &data_tmp) != EOF)
            hellfire_burn_speed_coeff[i] = data_tmp;
             fgets(descript, 80, fp);
             P(printf("hellfire_burn_speed_coeff(%3d) is%11.3f %s", i,
```

ı

```
hellfire_burn_speed_coeff[i], descript););
           ++i;
     }
     fclose(fp);
  END DEFAULT BURN SPEED DATA FOR miss_hellfr.c READ FROM FILE */
/* DEFAULT COAST SPEED DATA FOR miss_hellfr.c READ FROM FILE */
     fp = fopen("/simnet/data/ms_hf_cs.d", "r");
     if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_hf_cs.d\n");
            exit();
      }
      rewind(fp);
            Read degree of polynomial
                                           */
        fscanf(fp, "%d", &data_tmp_int);
      hellfr_miss_poly_deg[2] = data_tmp_int;
      fgets(descript, 80, fp);
      P(printf("hellfr_miss_poly_deg(2) is%3d %s",
                        hellfr_miss_poly_deg[2], descript););
      /*
            Read array data
      i=0;
      while(fscanf(fp, "%f", &data_tmp) != EOF)
            hellfire_coast_speed_coeff[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("hellfire_coast_speed_coeff(%3d) is%11.3f %s", i,
                         hellfire_coast_speed_coeff[i], descript););
            ++i;
      }
      fclose(fp);
/* END DEFAULT COAST SPEED DATA FOR miss_hellfr.c READ FROM FILE */
    mptr->state = FALSE;
    mptr->max_flight_time = HELLFIRE_MAX_FLIGHT_TIME;
    mptr->max_turn_directions = 1;
    speed_factor = MISSILE_US_SPEED_FACTOR;
    max_range_limit = MISSILE_US_MAX_RANGE_LIMIT;
    max_range_squared = max_range_limit * max_range_limit;
    hellfire_ammo_type = munition_US_Hellfire;
 }
 void missile_hellfire_set_speed_factor( scale_speed )
 REAL scale_speed;
 {
     speed_factor = scale_speed;
 }
```

```
void missile_hellfire_set_max_range_limit( limit_range )
REAL limit_range;
    max_range_limit = limit_range;
    max_range_squared = max_range_limit * max_range_limit;
 void missile_hellfire_set_ammo_type( ammo )
 ObjectType ammo;
    hellfire_ammo_type = ammo;
_ /**********************
  * ROUTINE: missile_hellfire_calc_tof
  * PARAMETERS: range - Range to target.
  * RETURNS: Time Of Flight for _range_ meters to target.
  * PURPOSE: This routine evaluates the TOF poly and returns *
               the time of flight for a Hellfire Missile
               to fly _range_ meters.
     **********************
 REAL missile_hellfire_calc_tof( range )
 REAL range;
    REAL time;
        missile_util_eval_poly( HELLFIRE_TOF_DEG, hellfire_tof_coeff, range );
     time =
     return( (time / speed_factor) );
 }
  /*****************************
   * ROUTINE: missile_hellfire_fire
   * PARAMETERS: mptr - A pointer to the HELLFIRE missile that
                   is to be launched.
            launch_point - The location in world
                          coordinates that the missile is *
                          launched from.
             launch_to_world - The transformation matrix of *
                            the launch platform to the
                             world.
             launch_speed - The speed of the launch
                          platform (assumed to be in the
                          direction of the missile). *
             tube - The tube the missile was launched from.
   * RETURNS: none
   * PURPOSE: This routine performs the functions
             specifically related to the firing of a
             Hellfire missile.
   **********
  void missile_hellfire_fire (mptr, launch_point, launch_to_world, launch_speed,
          tube)
```

```
MISSILE *mptr;
VECTOR launch_point;
T_MATRIX launch_to_world;
REAL launch_speed;
int tube;
{
/*/
 * Set the initial time, location, orientation, and speed of the generic
* missile.
/*/
#ifdef notdeff
    if( max_range_limit > 0.0 )
       mptr->max_flight_time =
           1.0 + missile_hellfire_calc_tof( max_range_limit );
#endif
   mptr->time = 0.0;
   vec_copy (launch_point, mptr->location);
   mat_copy (launch_to_world, mptr->orientation);
    mptr->speed = launch_speed +
       (speed_factor * (missile_util_eval_poly (HELLFIRE_BURN_SPEED_DEG,
                                               hellfire_burn_speed_coeff,
                                               0.0)));
   mptr->init_speed = launch_speed;
/*/
   Tell the rest of the world about the firing of the missile. If this
   cannot be done, return.
/*/
    if (!missile_util_comm_fire_missile (mptr, MSL_TYPE_MISSILE,
           map_get_ammo_entry_from_network_type (hellfire_ammo_type),
           hellfire_ammo_type, hellfire_ammo_type, NULL,
         targetUnknown, objectIrrelevant, tube))
       return;
 * If all was successful, set the missile state to TRUE and return.
    mptr->state = TRUE;
    return;
/**********************
 * ROUTINE: missile_hellfire_fly
                mptr - A pointer to the HELLFIRE missile that
 * PARAMETERS:
                  is to be flown out.
           target_location - The location in world
                            coordinates of the target.
 * RETURNS: none
 * PURPOSE: This routine performs the functions
           specifically related to the flying a HELLFIRE
           missile.
       ***********************
void missile_hellfire_fly (mptr, target_location)
MISSILE *mptr;
```

```
VECTOR target_location;
                                   /* The current time after launch (ticks). */
    register REAL time;
/*/
   Set and _time_. This is created mostly for increased readablity.
/*/
    time = mptr->time;
/*/
 * Find the current missile speed and the cosines of the maximum allowed turn
   angles in each direction. The equations used are different before and
 * after motor burnout.
/*/
    if (time < HELLFIRE_BURNOUT_TIME)</pre>
        mptr->speed = mptr->init_speed +
            (speed_factor *
             (missile_util_eval_poly (HELLFIRE_BURN_SPEED_DEG,
                                      hellfire_burn_speed_coeff, time) ));
    }
    else
        mptr->speed = mptr->init_speed +
            (speed_factor *
             (missile_util_eval_poly (HELLFIRE_COAST_SPEED_DEG,
                                      hellfire_coast_speed_coeff, time) ));
    }
/*/
 * Note that this is a temporary method of finding the max turn angle.
    mptr->cos_max_turn[0] = cos (sqrt (mptr->speed / SPEED_0) * THETA_0);
/*/
    If the missile is not armed, fly in a search trajectory; otherwise, fly
    in a targeted trajectory.
/*/
    if( max_range_limit > 0 &&
       kinematics_range_squared (veh_kinematics, mptr->location) >
       max_range_squared )
        missile_target_ground( mptr );
    else if (time < HELLFIRE_ARM_TIME)</pre>
        missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE, SIN_CLIMB,
             COS_CLIMB, SIN_LOCK, COS_LOCK, COS_TERM, COS_LOSE);
    else
        missile_target_agm (mptr, target_location, SIN_UNGUIDE, COS_UNGUIDE,
             SIN_CLIMB, COS_CLIMB, SIN_LOCK, COS_LOCK, COS_TERM, COS_LOSE);
 * Try to actually fly the missile. If this fails stop the missile altogether
 * and return.
    if (!missile_util_flyout (mptr))
        missile_hellfire_stop (mptr);
        return;
    }
    else
    {
```

```
/*/
       If the missile successfully flew, check for an intersection with the
       ground or a vehicle. If one is found, blow up the missile, stop its
       flyout and return.
/*/
       if (missile_util_comm_check_intersection (mptr, MSL_TYPE_MISSILE))
           missile_util_comm_check_detonate (mptr, MSL_TYPE_MISSILE);
           missile_hellfire_stop (mptr);
           return;
       }
   }
  If the missile is to continue to fly, return.
/*/
   return;
/***********************
 * ROUTINE: missile_hellfire_stop
 * PARAMETERS: mptr - A pointer to the HELLFIRE missile that *
                 is to be stopped.
 * RETURNS: none
 * PURPOSE: This routine causes all concerned to forget
           about the missile. It should be called when
           the flyout of any HELLFIRE missile is stopped
           (whether or not it has exploded). Note that
           this routine can only be called within this
           module.
 ************************************
static void missile_hellfire_stop (mptr)
MISSILE *mptr;
{
/*/
 * Tell the world to stop worrying about this missile then release the
 * memory for use by other missiles.
    missile_util_comm_stop_missile (mptr, MSL_TYPE_MISSILE);
    mptr->state = FALSE;
}
```

# Appendix H - Source code listing for miss\_kem.c.

The following appendix contains the source code listing for miss\_kem.c for convenience in document maintenance and understanding of the CSU.

# APPENDIX H - miss\_kem.c

```
/* $Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/libmissile/RCS/miss_kem
.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $ */
* $Log: miss_kem.c,v $
* Revision 1.4 1993/01/28 23:22:08 cm-adst
* P.DesMeules changes for spcr 31
* Revision 1.3 1993/01/06 21:13:01 cm-adst
 * R.Branson's changes for the weapons model.
* Revision 1.1 1992/09/30 16:39:52 cm-adst
 * Initial Version
*/
static char RCS_ID[] = "$Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/li
bmissile/RCS/miss_kem.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $";
* Revisions:
                                                       SP/CR Number
                                  Title
                Date
                         Author
     Version
                10/23/92 R. Branson Data File Initiali-
    1.2
                                     zation
                10/30/92 R. Branson Added pathname to data
    1.3
                                     directory
                11/25/92 R. Branson Changed %i to %d
     1.4
                 01/19/93 P.Desmeules Increased the size of the
                                                             31
     1.5
                                     fgets to make sure the
                                     whole line is read in.
     ****************
                    Description of Modification
       SP/CR No.
                     Hard coded defines changed to array elements.
                     Characteristics/parameter data array added.
                     Degree of polynomial data array added.
                     Added file reads for KEM characteristics/parameters,
                       burn speed coefficients, coast speed coefficients,
                       burn turn coefficients, and coast turn coeffi-
                       cients.
                     Added "/simnet/data/" to each data file pathname.
                 *******************
        ***************
  * FILE: miss_kem.c
```

```
* AUTHOR: Kris Bartol
* MAINTAINER: Kris Bartol: converted from miss_adat
             This file contains routines which fly out a
* PURPOSE:
             missile with the characteristics of a KEM
             missile.
* HISTORY: 10/23/90 kris: converted from miss_adat
* Copyright (c) 1989 BBN Systems and Technologies, Inc.
* All rights reserved.
**************************************
#include "stdio.h"
#include "math.h"
#include "sim_types.h"
#include "sim_dfns.h"
#include "basic.h"
#include "mun_type.h"
#include "libmap.h"
#include "libmatrix.h"
#include "miss_kem.h"
#include "libmiss_dfn.h"
#include "libmiss_loc.h"
* Debug macro
#ifdef FILEDBG
#define P(a)
#else
#define P(a)
#endif
 * Define missile characteristics.
#define KEM_MAX_FLIGHT_TIME kem_miss_char[1]
 * just after burnout, max V = ~3418 \text{ m/tick} = ~230 \text{ m/sec}
 * so in order to get the KEM missile to fly @ Vmax = 1524 m/2
 * must multiply the speed calculated by 6.\overline{626} ~= 1524 / 230
#define KEM_TO_MACH5_FACTOR kem_miss_char[2]
 * Define the states the _KEM_MISSILE_ can be in.
 #define KEM_FREE 0 /* No missile assigned. */
```

```
/st Missile flying and guided. st/
#define KEM_GUIDE 1
                        /* Missile flying but unguided. */
#define KEM_UNGUIDE 2
/*/
* The following terms set the order of the polynomials used to determine
* the speed or cosine of the maximum allowed turn rate of the missile
* at any point in time.
/*/
#define KEM_BURN_SPEED_DEG
                              kem_miss_poly_deg[0]
                             kem_miss_poly_deg[1]
#define KEM_COAST_SPEED_DEG
#define KEM_BURN_TURN_DEG
                              kem_miss_poly_deg[2]
                              kem_miss_poly_deg[3]
#define KEM_COAST_TURN_DEG
 * ADAT missile characteristic parameters initialized to default values.
/*/
static REAL kem_miss_char[10] =
             /* ticks (3.2 sec) */
    48.0,
             /* ticks (20.0 sec) */
 , 300.00,
            /* speed factor to raise from ADAT to KEM */
     6.626,
     0.0,
     0.0,
     0.0,
     0.0,
     0.0,
     0.0,
     0.0
};
 * The following are the default values of the degree of polynomials.
static int kem_miss_poly_deg[5] =
{
              /* Speed before motor burnout. */
   2,
              /* Speed after motor burnout. */
   4,
              /* Cosine of max turn before burnout. */
   3,
              /* Cosine of max turn after burnout. */
   5,
   0
};
 * Coefficients for the speed polynomial before motor burnout initialized
 * to default values.
 /*/
 static REAL kem_burn_speed_coeff[10] =
                          /* a_0 - m/tick */
     2.296,
                         /* a_1 - m/tick**2 */
     0.72990856,
                         /* a_2 - m/tick**3 */
     0.013310932,
```

```
0.0,
   0.0,
   0.0,
   0.0,
   0.0,
   0.0.
   0.0
};
* Coefficients for the speed polynomial after motor burnout.
static REAL kem_coast_speed_coeff[10] =
                        /* a_0 - m/tick */
  105.52162,
                        /* a_1 - m/tick**2 */
  -1.0157285,
                        /* a_2 - m/tick**3 */
   5.6124330e-3,
                       ./* a_3 - m/tick**4 */
   -1.6262608e-5,
                        /* a_4 - m/tick**5 */
    1.8991982e-8,
 0.0,
    0.0,
    0.0,
    0.0,
    0.0
};
 * Coefficients for the cosine of max turn polynomial before motor burnout.
/*/
static REAL kem_burn_turn_coeff[10] =
                         /* a_0 - cos(rad)/tick */
    0.999993,
                        /* a_1 - cos(rad)/tick**2 */
   -6.2386917e-7,
                        /* a_2 - cos(rad)/tick**3 */
    1.6146426e-7,
                        /* a_3 - cos(rad)/tick**4 */
   -9.720142e-7,
    0.0,
    0.0,
     0.0,
     0.0,
     0.0,
     0.0
 };
 * Coefficients for the cosine of max turn polynomial after motor burnout.
 /*/
 static REAL kem_coast_turn_coeff[10] =
                          /* a_0 - cos(rad)/tick */
     0.99753111,
                         /* a_1 - cos(rad)/tick**2 */
     5.5817986e-5,
                         /* a_2 - cos(rad)/tick**3 */
    -5.1276276e-7,
                         /* a_3 - cos(rad)/tick**4 */
     2.2388593e-9,
```

```
-5.1964622e-12, /* a_4 - cos(rad)/tick**5 */
4.5499104e-15, /* a_5 - cos(rad)/tick**6 */
   0.0,
   0.0,
   0.0,
   0.0
};
/*/
 * Memory for the missiles is declared in vehicle specific code. During
* initialization, a pointer is assigned to this memory then some memory
* issues are dealt with in this module.
/*/
static KEM_MISSILE *kem_array; /* A pointer to missile memory. */
                                 /* The number of defined missiles. */
static int num_kems;
 * Declare static functions.
/*/
static void missile_kem_stop ();
       ****************
 * ROUTINE: missile_kem_init
 * PARAMETERS: missile_array - A pointer to an array of *
                          KEM missiles defined in *
                          vehicle specific code.
           num_missiles - The number missiles defined in
                          _missile_array_.
 * RETURNS: none
 * PURPOSE: This routine copies the parameters into
       variables static to this module and initializes *
           the state of all the missiles.
 ***************************
void missile_kem_init (missile_array, num_missiles)
KEM_MISSILE missile_array[];
int num_missiles;
            /* A counter. */
    int i;
      int data_tmp_int;
      float data_tmp;
      char descript[80];
      FILE *fp;
      P(printf("$$$$$ KEM missile file data $$$$\n"););
 /* DEFAULT CHARACTERISTICS DATA FOR miss_kem.c READ FROM FILE
       fp = fopen("/simnet/data/ms_km_ch.d","r");
       if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_km_ch.d\n");
             exit();
```

```
}
 rewind(fp);
        Read array data
  i=0;
 while(fscanf(fp, "%f", &data_tmp) != EOF){
        kem_miss_char[i] = data_tmp;
        fgets(descript, 80, fp);
        P(printf("kem_miss_char(%3d) is%11.3f %s", i,
              kem_miss_char[i], descript););
        ++i;
  }
  fclose(fp);
END DEFAULT CHARACTERISTICS DATA FOR miss_kem.c READ FROM FILE
                                                                     */
                                                                */
DEFAULT BURN SPEED DATA FOR miss_kem.c READ FROM FILE
  fp = fopen("/simnet/data/ms_km_bs.d","r");
  if(fp==NULL){
        fprintf(stderr, "Cannot open /simnet/data/ms_km_bs.d\n");
        exit();
  }
  rewind(fp);
        Read degree of polynomial */
  fscanf(fp,"%d", &data_tmp_int);
  KEM_BURN_SPEED_DEG = data_tmp_int;
  fgets(descript, 80, fp);
  P(printf("kem_miss_poly_deg(0) is%3d %s",
         KEM_BURN_SPEED_DEG, descript););
         Read array data
                          */
   i=0;
   while(fscanf(fp, "%f", &data_tmp) != EOF){
         kem_burn_speed_coeff[i] = data_tmp;
         fgets(descript, 80, fp);
         P(printf("kem_burn_speed_coeff(%3d) is%11.3f %s", i,
                     kem_burn_speed_coeff[i], descript););
         ++i;
   }
   fclose(fp);
END DEFAULT BURN SPEED DATA FOR miss_kem.c READ FROM FILE
                                                                 */
                                                                 */
 DEFAULT COAST SPEED DATA FOR miss_kem.c READ FROM FILE
   fp = fopen("/simnet/data/ms_km_cs.d", "r");
   if(fp==NULL){
         fprintf(stderr, "Cannot open /simnet/data/ms_km_cs.d\n");
         exit();
   }
```

```
rewind(fp);
       Read degree of polynomial */
 fscanf(fp, "%d", &data_tmp_int);
 KEM_COAST_SPEED_DEG = data_tmp_int;
 fgets(descript, 80, fp);
 P(printf("kem_miss_poly_deg(1) is%3d %s",
       KEM_COAST_SPEED_DEG, descript););
        Read array data
 i=0;
 while(fscanf(fp, "%f", &data_tmp) != EOF){
        kem_coast_speed_coeff[i] = data_tmp;
        fgets(descript, 80, fp);
        P(printf("kem_coast_speed_coeff(%3d) is%11.3f %s", i,
                    kem_coast_speed_coeff[i], descript););
        ++i;
  }
  fclose(fp);
END DEFAULT COAST SPEED DATA FOR miss_kem.c READ FROM FILE
                                                                */
DEFAULT BURN TURN DATA FOR miss_kem.c READ FROM FILE
                                                                */
  fp = fopen("/simnet/data/ms_km_bt.d","r");
  if(fp==NULL){
        fprintf(stderr, "Cannot open /simnet/data/ms_km_bt.d\n");
        exit();
  }
  rewind(fp);
        Read degree of polynomial */
  fscanf(fp, "%d", &data_tmp_int);
  KEM_BURN_TURN_DEG = data_tmp_int;
  fgets(descript, 80, fp);
  P(printf("kem_miss_poly_deg(2) is%3d %s",
        KEM_BURN_TURN_DEG, descript););
        Read array data
  /*
  i=0:
  while(fscanf(fp, "%f", &data_tmp) != EOF){
         kem_burn_turn_coeff[i] = data_tmp;
         fgets(descript, 80, fp);
         P(printf("kem_burn_turn_coeff(%3d) is%11.3f %s", i,
                     kem_burn_turn_coeff[i], descript););
         ++i;
   }
   fclose(fp);
END DEFAULT BURN TURN DATA FOR miss_kem.c READ FROM FILE
                                                                 */
```

```
/* DEFAULT COAST TURN DATA FOR miss_kem.c READ FROM FILE
                                                                 */
      fp = fopen("/simnet/data/ms_km_ct.d","r");
      if(fp==NULL){
           fprintf(stderr, "Cannot open /simnet/data/ms_km_ct.d\n");
            exit();
      }
      rewind(fp);
            Read degree of polynomial */
      fscanf(fp, "%d", &data_tmp_int);
      KEM_COAST_TURN_DEG = data_tmp_int;
      fgets(descript, 80, fp);
      P(printf("kem_miss_poly_deg(3) is%3d %s",
            KEM_COAST_TURN_DEG, descript););
            Read array data */
      i=0;
      while(fscanf(fp, "%f", &data_tmp) != EOF){
            kem_coast_turn_coeff[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("kem_coast_turn_coeff(%3d) is%11.3f %s", i,
                        kem_coast_turn_coeff[i], descript););
            ++i;
      }
      fclose(fp);
    END DEFAULT COAST TURN DATA FOR miss_kem.c READ FROM FILE
    num_kems = num_missiles;
    kem_array = missile_array;
     for (i = 0; i < num_missiles; i++)</pre>
        kem_array[i].mptr.state = KEM_FREE;
        kem_array[i].mptr.max_flight_time = KEM_MAX_FLIGHT_TIME;
        kem_array[i].mptr.max_turn_directions = 1;
 }
 int missile_kem_is_free( missile )
 int missile;
 {
     return( (kem_array[missile].mptr.state == KEM_FREE ));
            ***********
  * ROUTINE: missile_kem_fire
                   kptr - A pointer to the KEM missile to be *
  * PARAMETERS:
                    fired.
             launch_point - The location in world
                            coordinates that the missile is *
```

```
launched from.
           loc_sight_to_world - The sight to world
                                transformation matrix used *
                                only in this routine.
           launch_speed - The speed of the launch
                          platform (assumed to be in the
                          direction of the missile). *
               target_id - Target's tracking ID
               target_loc - location of target in World Coord
           target_vehicle_id - The vehicle ID of the *
                              target (if any).
* RETURNS: TRUE if successful, FALSE if not.
  PURPOSE: This routine performs the functions
           specifically related to the firing of a KEM
       ******
int missile_kem_fire (kptr, launch_point, loc_sight_to_world, launch_speed,
                     target_id, target_loc, target_vehicle_id)
KEM_MISSILE *kptr;
VECTOR launch_point;
T_MATRIX loc_sight_to_world;
REAL launch_speed;
int target_id;
VECTOR target_loc;
VehicleID *target_vehicle_id;
                              /* A counter. */
    int i;
                              /* Pointer to the particular generic missile
    MISSILE *mptr;
                                 pointed at by _kptr_. */
                              /* Indication of whether target is known. */
    int comm_target_type;
    Find _mptr_ and _target_id_.
    mptr = &(kptr->mptr);
    if (target_vehicle_id == 0)
        kptr->target_vehicle_id.vehicle = vehicleIrrelevant;
        kptr->target_vehicle_id = *target_vehicle_id;
    kptr->target_id = target_id;
    vec_copy( target_loc, kptr->target_pos );
 * Set the initial time, location, orientation, and speed of the generic
   missile.
    mptr->time = 0.0;
    vec_copy (launch_point, mptr->location);
    mat_copy (loc_sight_to_world, mptr->orientation);
    mptr->speed = (missile_util_eval_poly (KEM_BURN_SPEED_DEG,
            kem_burn_speed_coeff, 0.0) * KEM_TO_MACH5_FACTOR) + launch_speed;
    mptr->init_speed = launch_speed;
```

```
if (kptr->target_vehicle_id.vehicle == vehicleIrrelevant)
      comm_target_type = targetUnknown;
  else
      comm_target_type = targetIsVehicle;
* Tell the rest of the world about the firing of the missile. If this
  cannot be done, return FALSE.
   if (!missile_util_comm_fire_missile (mptr, MSL_TYPE_MISSILE,
          map_get_ammo_entry_from_network_type (munition_US_ADATS),
          munition_US_ADATS, munition_US_ADATS, &(kptr->target_vehicle_id),
          comm_target_type, objectIrrelevant, 0 /*tube*/))
       return (FALSE);
  If all was successful, fly missile in guided state.
   mptr->state = KEM_GUIDE;
   return (TRUE);
}
/************************
 * ROUTINE: missile_kem_update_guidance
 * PARAMETERS: missile - An index to the KEM missile that
                is to be updated.
          target_location - The location in world
                      coordinates of the target
 * RETURNS: none
 * PURPOSE: This routine updates the KEM's target's
          position in world coordinates.
 *************
void missile_kem_update_guidance( missile, target_location )
int missile;
VECTOR target_location;
    if( kem_array[missile].mptr.state == KEM_GUIDE )
       vec_copy( target_location, kem_array[missile].target_pos );
/***********************
 * ROUTINE: missile_kem_fly
 * PARAMETERS: missile - An index to the KEM missile that
                 is to be flown out.
 * RETURNS: none
 * PURPOSE: This routine performs the functions
           specifically related to the flying a KEM *
           missile.
     ***********************************
 void missile_kem_fly( missile )
 int missile;
```

```
{
                          /* A pointer to a KEM missile */
   KEM_MISSILE *kptr;
                         /* A pointer to the generic aspects of _kptr_. */
   MISSILE *mptr;
                         /* The current time after launch (ticks). */
   REAL time;
   Set _kptr_, _mptr_ and _time_. These values are created mostly
   for increased readablity.
   kptr = &kem_array[missile];
    mptr = &(kptr->mptr);
   time = mptr->time;
 * Find the current missile speed and the cosines of the maximum allowed turn
 * angles in each direction. The equations used are different before and
   after motor burnout.
 */
    if (time < KEM_BURNOUT_TIME)</pre>
        mptr->speed = (missile_util_eval_poly (KEM_BURN_SPEED_DEG,
                kem_burn_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
                    mptr->init_speed;
        mptr->cos_max_turn[0] = missile_util_eval_poly (KEM_BURN_TURN_DEG,
                kem_burn_turn_coeff, time);
    }
    else
    {
        mptr->speed = (missile_util_eval_poly (KEM_COAST_SPEED_DEG,
                kem_coast_speed_coeff, time) * KEM_TO_MACH5_FACTOR) +
                    mptr->init_speed;
        mptr->cos_max_turn[0] = missile_util_eval_poly (KEM_COAST_TURN_DEG,
                kem_coast_turn_coeff, time);
    }
    Find the target point = Missile's Target's position regardless of state
    if( mptr->state == KEM_GUIDE || mptr->state == KEM_UNGUIDE )
        missile_target_point( mptr, kptr->target_pos );
    else
        printf ("MISSILE_KEM: disallowed missile state %d\n", mptr->state);
    Try to actually fly the missile. If this fails stop the missile altogether
    and return.
  */
    if (!missile_util_flyout (mptr)) /* checks for time > max_flight_time */
        missile_kem_stop (kptr);
        return;
     else
     {
         If the missile successfully flew, check for an intersection with the
         ground or a vehicle. If one is found, blow up the missile, stop its
         flyout and return.
  */
```

```
if (missile_util_comm_check_detonate (mptr, MSL_TYPE_MISSILE))
           missile_kem_stop (kptr);
           return;
       }
   }
* If the missile is to continue to fly, return.
   return;
}
 * ROUTINE: missile_kem_reset_missiles
 * PARAMETERS: none
 * RETURNS: none
* PURPOSE: This routine puts any flying missile into an
           unguided state.
void missile_kem_reset_missiles ()
{
    int i;
 * Reset all flying missiles.
    for (i = 0; i < num\_kems; i++)
        if( kem_array[i].mptr.state == KEM_GUIDE )
            kem_array[i].mptr.state = KEM_UNGUIDE;
}
 * ROUTINE: missile_kem_stop
 * PARAMETERS: kptr - A pointer to the KEM missile that is to *
                  be stopped.
  * RETURNS: none
  * PURPOSE: This routine causes all concerned to forget
            about the missile. It should be called when
            the flyout of any KEM missile is stopped *
            (whether or not it has exploded). Note that
            this routine can only be called within this
            module.
        *************************************
 static void missile_kem_stop (kptr)
 KEM_MISSILE *kptr;
 {
 /*/
  * Tell the world to stop worrying about this missile then release the
  * memory for use by other missiles.
 /*/
```

```
missile_util_comm_stop_missile (&(kptr->mptr), MSL_TYPE_MISSILE);
kptr->mptr.state = KEM_FREE;
```

}

# Appendix I - Source code listing for miss\_maverck.c.

The following appendix contains the source code listing for miss\_maverck.c for convenience in document maintenance and understanding of the CSU.

```
/* $Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/libmissile/RCS/miss_mav
erck.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $ */
 * $Log: miss_maverck.c,v $
 * Revision 1.4 1993/01/28 23:22:08 cm-adst
 * P.DesMeules changes for spcr 31
 * Revision 1.3 1993/01/06 21:13:31 cm-adst
 * R.Branson's changes for the weapons model.
 * Revision 1.1 1992/09/30 16:39:52 cm-adst
 * Initial Version
static char RCS_ID[] = "$Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/li
bmissile/RCS/miss_maverck.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $";
/*******************************
  Revisions:
                                                             SP/CR Number
                                       Title
                            Author
                  Date
     Version
                  10/23/92 R. Branson Data File Initiali-
      1.2
                                         zation
                  10/30/92 R. Branson Added pathname to data
     1.3
                                         directory
                   11/25/92 R. Branson Changed %i to %d
     1.4
                   01/19/93 P.Desmeules Increased the size of the
      1.5
                                         fgets to make sure the
                                         whole line is read in.
                     Description of Modification
        SP/CR No.
                       Hard coded defines changed to array elements.
                       Characteristics/parameter data array added.
                       Degree of polynomial data array added.
                       Added file reads for maverick characteristics/
                          parameters, burn speed coefficients, and coast
                          speed coefficients.
                       Added "/simnet/data/" to each data file pathname.
          ******************************
             miss_maverick.c
  * FILE:
              Bryant Collard
  * AUTHOR:
```

```
* MAINTAINER: Bryant Collard
             This file contains routines which fly out a
* PURPOSE:
             missile with the characteristics of a MAVERICK
             missile.
             12/8/88 bryant: Creation
* HISTORY:
             4/24/89 bryant: Added static memory allocation.
             7/26/91 carol : libtrack/intervis integration
* Copyright (c) 1988 BBN Systems and Technologies, Inc.
* All rights reserved.
#include "stdio.h"
#include "math.h"
#include "sim_types.h"
#include "sim_dfns.h"
#include "basic.h"
#include "mun_type.h"
#include "libmap.h"
#include "libmatrix.h"
#include "libnear.h"
#include "libtrack.h"
#include "miss_maverck.h"
#include "libmiss_dfn.h"
#include "libmiss_loc.h"
* Debug macro
#ifdef FILEDBG
#define P(a)
#else
#define P(a)
#endif
 * Define missile characteristics.
/*/
                                 maverick_miss_char[ 0]
#define MAVERICK_ARM_TIME
#define MAVERICK_BURNOUT_TIME
                                 maverick_miss_char[ 1]
#define MAVERICK_MAX_FLIGHT_TIME maverick_miss_char[ 2]
#define MAVERICK_LOCK_THRESHOLD maverick_miss_char[ 3]
#define MAVERICK_HOLD_THRESHOLD maverick_miss_char[ 4]
                                maverick_miss_char[ 5]
 #define SPEED_0
                                maverick_miss_char[ 6]
 #define THETA_0
  * Set parmeters which will control flight trajectory behavior.
 /*/
```

```
maverick_miss_char[ 7]
#define SIN_UNGUIDE
                             maverick_miss_char[ 8]
#define COS_UNGUIDE
                             maverick_miss_char[ 9]
#define SIN_CLIMB
                             maverick_miss_char[10]
#define COS_CLIMB
                             maverick_miss_char[11]
#define SIN_LOCK
                            maverick_miss_char[12]
#define COS_LOCK
                            maverick_miss_char[13]
#define COS_TERM
                             maverick_miss_char[14]
#define COS_LOSE
 * Define the states the _MAVERICK_MISSILE_ can be in.
                            /* No missile assigned. */
/* Missile assigned to ready state. */
/* Missile assigned to ready state. */
#define MAVERICK_FREE 0
#define MAVERICK_READY 1
                              /* Missile assigned to flying state. */
#define MAVERICK_FLYING 2
/*/
 * The following terms set the order of the polynomials used to determine
 * the speed or cosine of the maximum allowed turn rate of the missile
 * at any point in time.
/*/
#define MAVERICK_BURN_SPEED_DEG maverick_miss_poly_deg[0]
#define MAVERICK_COAST_SPEED_DEG maverick_miss_poly_deg[1]
 * Maverick missile characteristic parameters initialized to default values.
/*/
 static REAL maverick_miss_char[15] =
                  /* maverick arm time ticks (1.3 sec) */
    20.0,
                   /* maverick burnout time ticks (1.5 sec) */
   22.5.
                   /* maverick max flight time ticks (60 sec) */
   900.0,
     0.989073800, /* maverick lock threshold cos (6 deg) ** 2 */
     0.969846310, /* maverick hold threshold cos (10 deg) ** 2 */
    28.33333333, /* speed_0
                                 */
     0.046542113, /* theta_0
                  /* sin level unguided flight. */
     0.0,
               /* cos level unguided flight. */
     1.0,
     0.004072424, /* sin climb 3.5 deg/sec */
     0.999991708, /* cos climb 3.5 deg/sec */
0.087155743, /* sin lock 5 deg */
     0.996194698, /* cos lock 5 deg */
     0.173648178, /* cos terminal 80 deg */
     0.939692621 /* cos loose lock 20 deg */
 };
 /*/
  * The following terms set the order of the polynomials used to determine
  * the speed.
 /*/
 static int maverick_miss_poly_deg[2] =
 {
             /* Maverick burn speed degree. */
     1,
```

```
/* Maverick coast speed degree. */
};
* Coefficients for the speed polynomial before motor burnout.
/*/
static REAL maverick_burn_speed_coeff[5] =
                      /* a_0 - m/tick (67.0 m/sec) */
    0.03333333,
                      /* a_1 - m/tick**2 (274.9732662 m/sec**2) */
    1.25777777
};
 * Coefficients for the speed polynomial after motor burnout.
static REAL maverick_coast_speed_coeff[5] =
                        /* a_0 - m/tick (327.2858074 m/sec) */
                        /* a_1 - m/tick**2 (-21.4609544 m/sec**2) */
    30.46972849,
    -9.7721160e-2,
                        /* a_2 - m/tick**3 (0.8227650 m/sec**3) */
     1.2433925e-4,
                        /* a_3 - m/tick**4 (-0.0133200 m/sec**4) */
    -5.4061501e-8
 };
  * Memory for the missiles is declared in vehicle specific code. During
 /*/
  * initialization, a pointer is assigned to this memory then all memory
  * issues are dealt with in this module.
 /*/
 static MAVERICK_MISSILE *maverick_array; /* A pointer to missile memory. */
                                         /* The number of defined missiles. */
 static int num_mavericks;
 #define STRING_LEN 20
 static char prelaunch_intervis_method [STRING_LEN + 1] = "lrf";
 static char in_flight_intervis_method [STRING_LEN + 1] = "omniscient";
 static PFI pel_callback_func;
 static REAL maverick_cone_threshold;
  * Declare static functions.
  /*/
 static void missile_maverick_fly ();
 static MAVERICK_MISSILE *missile_maverick_get_missile_from_sensor_id ();
  static void missile_maverick_lock_handler ();
  static void missile_maverick_break_lock_handler ();
  static REAL missile_maverick_detectibility ();
  static void missile_maverick_object_update ();
  /***********************
   * ROUTINE: missile_maverick_init
   * PARAMETERS: missile_array - A pointer to an array of *
```

```
MAVERICK missiles defined in
                          vehicle specific code.
           num_missiles - The number missiles defined in
                         _missile_array_.
* RETURNS: none
* PURPOSE: This routine copies the parameters into
           variables static to this module and initializes *
           the state of all the missiles.
 void missile_maverick_init (missile_array, num_missiles, func)
MAVERICK_MISSILE missile_array[];
int num_missiles;
PFI func;
            /* A counter. */
    int i;
           data_tmp_int;
      int
      float data_tmp;
      char descript[80];
      FILE *fp;
      P(printf("$$$$ MAVERICK missile file data $$$$\n"););
                                                                            */
/* DEFAULT CHARACTERISTICS DATA FOR miss_maverck.c READ FROM FILE
      fp = fopen("/simnet/data/ms_mk_ch.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_mk_ch.d\n");
            exit();
      }
      rewind(fp);
            Read array data
       /*
      i=0;
      while(fscanf(fp, "%f", &data_tmp) != EOF){
            maverick_miss_char[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("maverick_miss_char(%3d) is%11.3f %s", i,
                  maverick_miss_char[i], descript););
            ++i;
       }
 /* END DEFAULT CHARACTERISTICS DATA FOR miss_maverck.c READ FROM FILE */
       fclose(fp);
                                                                       */
 /* DEFAULT BURN SPEED DATA FOR miss_maverck.c READ FROM FILE
       fp = fopen("/simnet/data/ms_mk_bs.d","r");
       if(fp==NULL){
             fprintf(stderr, "Cannot open /simnet/data/ms_mk_bs.d\n");
             exit();
       }
       rewind(fp);
```

```
Read degree of polynomial */
    fscanf(fp, "%d", &data_tmp_int);
    MAVERICK_BURN_SPEED_DEG = data_tmp_int;
    fgets(descript, 80, fp);
    P(printf("maverick_miss_poly_deg(0) is%3d %s",
          MAVERICK_BURN_SPEED_DEG, descript););
          Read array data */
    i=0:
    while(fscanf(fp, "%f", &data_tmp) != EOF){
          maverick_burn_speed_coeff[i] = data_tmp;
           fgets(descript, 80, fp);
          P(printf("maverick_burn_speed_coeff(%3d) is%11.3f %s", i,
                       maverick_burn_speed_coeff[i], descript););
          ++i;
    }
     fclose(fp);
  END DEFAULT BURN SPEED DATA FOR miss_maverck.c READ FROM FILE */
  DEFAULT COAST SPEED DATA FOR miss_maverck.c READ FROM FILE
                                                                        */
     fp = fopen("/simnet/data/ms_mk_cs.d", "r");
     if(fp==NULL){
           fprintf(stderr, "Cannot open /simnet/data/ms_mk_cs.d\n");
           exit();
     }
     rewind(fp);
           Read degree of polynomial */
     fscanf(fp, "%d", &data_tmp_int);
     MAVERICK_COAST_SPEED_DEG = data_tmp_int;
     fgets(descript, 80, fp);
     P(printf("maverick_miss_poly_deg(1) is%3d %s",
           MAVERICK_COAST_SPEED_DEG, descript););
           Read array data
     i=0;
     while(fscanf(fp, "%f", &data_tmp) != EOF){
           maverick_coast_speed_coeff[i] = data_tmp;
            fgets(descript, 80, fp);
           P(printf("maverick_coast_speed_coeff(%3d) is%11.3f %s", i,
                        maverick_coast_speed_coeff[i], descript););
            ++i;
     }
     fclose(fp);
/* END DEFAULT COAST SPEED DATA FOR miss_maverck.c READ FROM FILE
                                                                         */
   maverick_cone_threshold = MAYERICK_LOCK_THRESHOLD;
```

```
num_mavericks = num_missiles;
   maverick_array = missile_array;
   for (i = 0; i < num_missiles; i++)</pre>
     maverick_array[i].mptr.state = MAVERICK_FREE;
       maverick_array[i].mptr.max_flight_time = MAVERICK_MAX_FLIGHT_TIME;
       maverick_array[i].mptr.max_turn_directions = 1;
     maverick_array[i].object_being_tracked = NO_OBJECT;
     maverick_array[i].sensor_id = NULL;
   pel_callback_func = func;
}
/**************
 * ROUTINE: missile_maverick_sensor_init
 * PARAMETERS:
               none
 * RETURNS: none
 * PURPOSE: Calls to initialize a libtrack sensor
void missile_maverick_sensor_init (mvptr, iv_method)
MAVERICK_MISSILE *mvptr;
char *iv_method;
    if (TrackSensorInit (missile_maverick_lock_handler,
                 missile_maverick_break_lock_handler,
                 missile_maverick_detectibility,
                 pel_callback_func,
                 missile_maverick_object_update,
                 E_NANO,
                  &mvptr -> sensor_id) < 0)
        printf ("missile_maverick_sensor_init: TrackSensorInit: %s\n",
            TrackErrString ());
    if (TrackSetIntervisibility (mvptr -> sensor_id, prelaunch_intervis_method) <</pre>
0)
        printf ("missile_maverick_sensor_init: TrackSetIntervisibility: %s\n",
            TrackErrString ());
    if (TrackSetPersistence (mvptr -> sensor_id, 5 /* ticks of persistence */)
        printf ("missile_maverick_sensor_init: TrackSetPersistence: %s\n",
            TrackErrString ());
    if (TrackSetMaxResponses (mvptr -> sensor_id, 1) < 0)</pre>
        printf ("missile_maverick_sensor_init: TrackSetMaxResponses: %s\n",
            TrackErrString ());
     if (TrackSetVehicleID (mvptr -> sensor_id, network_get_vehicle_id ()) < 0)</pre>
        printf ("missile_maverick_sensor_init: TrackSetVehicleID: %s\n",
            TrackErrString ());
 }
```

```
********
* ROUTINE: missile_maverick_ready
               none
* PARAMETERS:
* RETURNS: A pointer to a missile that is currently *
           available.
* PURPOSE: This routine finds, if possible, a missile that *
           is not being used, puts it in a ready state and *
           returns a pointer to it.
 *****************************
MAVERICK_MISSILE *missile_maverick_ready ()
   int i; /* A counter. */
 * Try to find a free missile.
    for (i = 0; i < num_mavericks; i++)
/,*/
*
       If a free missile is found, put it in a ready state, clear the target
       ID and return a pointer to it.
/*/
       if (maverick_array[i].mptr.state == MAVERICK_FREE)
           maverick_array[i].mptr.state = MAVERICK_READY;
           maverick_array[i].target_vehicle_id.vehicle = vehicleIrrelevant;
         missile_maverick_sensor_init (&maverick_array[i],
                              prelaunch_intervis_method);
           return (&maverick_array[i]);
       }
    }
/*/
 * If no free missile is found, return a NULL pointer.
/*/
    return (NULL);
}
              **************
 * ROUTINE: missile_maverick_pre_launch
                 mvptr - A pointer to the missile that is to be *
 * PARAMETERS:
                   serviced.
            launch_point - The location of the missile in
                          world coordinates.
            launch_to_world - The transformation matrix of *
                             the missile to the world.
               veh_list - Vehicle list ID.
  * RETURNS: none
  * PURPOSE: This routine is called after a missile has been *
            readied and before it has been launched. It
            determines if the seeker head can see a target *
            and, if it can see a target, stores its
```

```
position.
void missile_maverick_pre_launch (mvptr, launch_point, launch_to_world,
    veh list)
MAVERICK_MISSILE *mvptr;
VECTOR launch_point;
T_MATRIX launch_to_world;
int veh_list;
    register TObjectP object;
    VECTOR object_loc;
 * tick libtrack to update location and see if any callbacks need to be
 * invoked.
    if (TrackUpdate (mvptr -> sensor_id, veh_list, launch_point,
                 launch_to_world[1]) < 0)</pre>
        printf ("missile_maverick_pre_launch: TrackUpdate: %s\n",
            TrackErrString ());
    If a target is found, store its location.
    if ((object = mvptr -> object_being_tracked) != NO_OBJECT)
        mvptr->target_vehicle_id = object -> var.vehicleID;
      GetLocationOfTObject (object, object_loc);
/* change pursuit to take a VECTOR rather than VAP for location */
        missile_target_pursuit (&(mvptr->mptr), object_loc);
    }
    else
        mvptr->target_vehicle_id.vehicle = vehicleIrrelevant;
      if (TrackAcquire (mvptr -> sensor_id, veh_list, launch_point,
                     launch_to_world[1]) < 0)</pre>
          printf ("missile_maverick_pre_launch: TrackAcquire: %s\n",
                 TrackErrString ());
     }
 }
  * ROUTINE: missile_maverick_fire
                   mvptr - A pointer to the MAVERICK missile that *
  * PARAMETERS:
                     is to be launched.
             launch_point - The location in world
                            coordinates that the missile is *
                            launched from.
             launch_to_world - The transformation matrix of *
                               the launch platform to the
                               world.
             launch_speed - The speed of the launch
                            platform (assumed to be in the
                            direction of the missile). *
```

```
tube - The tube the missile was launched from.
  RETURNS: TRUE for a successful launch and FALSE for an
           unsuccessful launch.
 * PURPOSE: This routine performs the functions
            specifically related to the firing of a
           MAVERICK missile.
int missile_maverick_fire (mvptr, launch_point, launch_to_world, launch_speed,
        tube)
MAVERICK_MISSILE *mvptr;
VECTOR launch_point;
T_MATRIX launch_to_world;
REAL launch_speed;
int tube;
                           /* Pointer to the particular generic missile
    MISSILE *mptr;
                              pointed at by _mvptr_. */
   Get a pointer to the generic elements of the MAVERICK missile. This
    improves code readability.
    mptr = &(mvptr->mptr);
/*/
   Set the initial time, location, orientation, and speed of the generic
   missile.
/*/
    mptr->time = 0.0;
    vec_copy (launch_point, mptr->location);
    mat_copy (launch_to_world, mptr->orientation);
    mptr->speed = missile_util_eval_poly (MAVERICK_BURN_SPEED_DEG,
            maverick_burn_speed_coeff, 0.0) + launch_speed;
    mptr->init_speed = launch_speed;
 * Tell the rest of the world about the firing of the missile. If this
   cannot be done, release the missile memory and return FALSE.
/*/
    if (!missile_util_comm_fire_missile (mptr, MSL_TYPE_MISSILE,
            map_get_ammo_entry_from_network_type (munition_US_Maverick),
            munition_US_Maverick, munition_US_Maverick,
            &(mvptr->target_vehicle_id), targetIsVehicle, objectIrrelevant,
            tube))
    {
        mptr->state = MAVERICK_FREE;
        return (FALSE);
 /*/
    If all was successful, set the missile state to MAVERICK_FLYING and
    return TRUE.
 /*/
    mptr->state = MAVERICK_FLYING;
    return (TRUE);
 }
```

```
/************************
* ROUTINE: missile_maverick_fly_missiles
* PARAMETERS: veh_list - Vehicle list ID.
* RETURNS: none
* PURPOSE: This routine flies out all missiles in a *
         flying state.
**********
void missile_maverick_fly_missiles (veh_list)
int veh_list;
{
   /*/
  Fly out all flying missiles.
   for (i = 0; i < num_mavericks; i++)</pre>
      if (maverick_array[i].mptr.state == MAVERICK_FLYING)
          missile_maverick_fly (&(maverick_array[i]), veh_list);
   }
}
/************************
 * ROUTINE: missile_maverick_fly
 * PARAMETERS: mvptr - A pointer to the MAVERICK missile that *
                is to be flown out.
             veh_list - Vehicle list ID.
 * RETURNS: none
 * PURPOSE: This routine performs the functions
          specifically related to the flying a MAVERICK *
          missile.
 *******************************
static void missile_maverick_fly (mvptr, veh_list)
MAVERICK_MISSILE *mvptr;
int veh_list;
    register MISSILE *mptr; /* A pointer to the generic aspects of
                                 _mvptr_. */
                              /* The current time after launch (ticks). */
    REAL time;
                              /* The location of the target. */
    VECTOR target_location;
 /*/
 * Set _mptr_ and _time_. These values are created mostly for increased
 * readablity.
    mptr = &(mvptr->mptr);
    time = mptr->time;
 /*/
  * Find the current missile speed and the cosine of the maximum allowed turn
  * angle. The equations used are different before and after motor burnout.
```

```
/*/
    if (time < MAVERICK_BURNOUT_TIME)</pre>
        mptr->speed = missile_util_eval_poly (MAVERICK_BURN_SPEED_DEG,
                maverick_burn_speed_coeff, time) + mptr->init_speed;
    else
        mptr->speed = missile_util_eval_poly (MAVERICK_COAST_SPEED_DEG,
                maverick_coast_speed_coeff, time) + mptr->init_speed;
    }
/*/
    Note that this is a temporary method of finding turn angle.
    mptr->cos_max_turn[0] = cos (sqrt (mptr->speed / (SPEED_0 +
            mptr->init_speed)) * THETA_0);
    if (TrackUpdate (mvptr -> sensor_id, veh_list, mptr -> location,
                 mptr -> orientation[1]) < 0)</pre>
        printf ("missile_maverick_fly: TrackUpdate: %s\n", TrackErrString ());
/1*/
    Find the target point to which the missile is to fly. The missile ignores
    any targets until it is armed.
/*/
    if (time < MAVERICK_ARM_TIME)</pre>
        missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE, SIN_CLIMB,
                 COS_CLIMB, SIN_LOCK, COS_LOCK, COS_TERM, COS_LOSE);
    else
      TObjectP object = mvptr -> object_being_tracked;
/*/
        Try to find a target. If one is found, fly towards it in the
        proper trajectory, otherwise, fly in a search trajectory.
 /*/
         if (object != NO_OBJECT)
         {
           VECTOR target_location;
           GetLocationOfTObject (object, target_location);
             mvptr->target_vehicle_id = object -> var.vehicleID;
             missile_target_agm (mptr, target_location, SIN_UNGUIDE,
                     COS_UNGUIDE, SIN_CLIMB, COS_CLIMB, SIN_LOCK, COS_LOCK,
                     COS_TERM, COS_LOSE);
         }
         else
             mvptr->target_vehicle_id.vehicle = vehicleIrrelevant;
           if (TrackAcquire (mvptr -> sensor_id, veh_list, mptr -> location,
                         mptr -> orientation[1]) < 0)</pre>
               printf ("missile_maverick_fly: TrackAcquire: %s\n",
                   TrackErrString ());
             missile_target_agm (mptr, NULL, SIN_UNGUIDE, COS_UNGUIDE,
                     SIN_CLIMB, COS_CLIMB, SIN_LOCK, COS_LOCK, COS_TERM,
                      COS_LOSE);
         }
     }
```

```
* Try to actually fly the missile. If this fails stop the missile altogether
* and return.
/*/
   if (!missile_util_flyout (mptr))
       missile_maverick_stop (mvptr);
       return;
   }
   else
/*/
       If the missile successfully flew, check for an intersection with the
       ground or a vehicle. If one is found, blow up the missile, stop its
       flyout and return.
/*/
       if (missile_util_comm_check_intersection (mptr, MSL_TYPE_MISSILE))
           missile_util_comm_check_detonate (mptr, MSL_TYPE_MISSILE);
           missile_maverick_stop (mvptr);
           return;
     . }
   }
/*/
 * If the missile is to continue to fly, return.
/*/
    return;
     ***********
 * ROUTINE: missile_maverick_stop
 * PARAMETERS: mvptr - A pointer to the MAVERICK missile that *
                 is to be stopped.
 * RETURNS: none
 * PURPOSE: This routine causes all concerned to forget
           about the missile. It should be called when
           the flyout of any MAVERICK missile is stopped
           (whether or not it has exploded).
 ************************************
void missile_maverick_stop (mvptr)
MAVERICK_MISSILE *mvptr;
 {
  * If the world has been told to worry about this missile, tell it to stop
  * then release missile memory for use by other missiles.
 /*/
     if (mvptr->mptr.state == MAVERICK_FLYING)
        missile_util_comm_stop_missile (&(mvptr->mptr), MSL_TYPE_MISSILE);
     mvptr->mptr.state = MAVERICK_FREE;
     TrackSensorUnInit (mvptr -> sensor_id);
     mvptr -> sensor_id = NULL;
     mvptr -> object_being_tracked = NO_OBJECT; /* perhaps call break lock? */
```

```
}
static MAVERICK_MISSILE *missile_maverick_get_missile_from_sensor_id (sensor_id)
int sensor_id;
    register MAVERICK_MISSILE *mvptr = maverick_array;
    register int i;
    for (i = 0; i < num_mavericks; i++, mvptr++)</pre>
      if (mvptr -> sensor_id == sensor_id)
          return (mvptr);
    }
    return (NULL);
}
static void missile_maverick_lock_handler (sensor_id, object)
int sensor_id;
TObjectP object;
    MAVERICK_MISSILE *mvptr;
    if (object == NO_OBJECT)
        if (TrackDontLock (sensor_id, object) < 0)</pre>
          printf ("MaverickLockHandler: TrackDontLock: %s\n",
                 TrackErrString ());
      return;
    }
    if ((mvptr = missile_maverick_get_missile_from_sensor_id (sensor_id))
       != NULL)
/* already tracking an object, but because of the delay from the TrackAqcuire
   call, the lock handler has been invoked again. It does not matter if it is
   the same object or not as before. Just do not lock again */
       if (mvptr -> object_being_tracked != NO_OBJECT)
           if (TrackDontLock (sensor_id, object) < 0)</pre>
               printf ("MaverickLockHandler: TrackDontLock: %s\n",
                   TrackErrString ());
           return;
       }
       mvptr -> object_being_tracked = object;
       if (TrackLock (sensor_id, object) < 0)</pre>
           printf ("MaverickLockHandler: TrackLock: %s\n", TrackErrString ());
     }
     else
         printf ("LockHandler: No missile for SensorId %d\n", sensor_id);
       if (TrackDontLock (sensor_id, object) < 0)</pre>
```

```
printf ("MaverickLockHandler: TrackDontLock: %s\n",
            TrackErrString ());
    }
}
static void missile_maverick_break_lock_handler (sensor_id, object)
int sensor_id;
TObjectP object;
    register MAVERICK_MISSILE *mvptr;
    if (object == NO_OBJECT)
        return;
    if ((mvptr = missile_maverick_get_missile_from_sensor_id (sensor_id))
      != NULL)
      if (mvptr -> object_being_tracked == NO_OBJECT)
            printf ("MaverickBreakLockHandler: BREAK LOCK BUT NOT LOCKED !!!\n");
          return;
      }
      if (mvptr -> object_being_tracked != object)
          printf ("MaverickBreakLockHandler: BREAK LOCK ON UNKNOWN OBJECT!!!\n");
          return;
      }
      if (TrackBreakLock (sensor_id, object) < 0)</pre>
          printf ("MaverickBreakLockHandler: TrackBreakLock: %s\n",
                 TrackErrString ());
      mvptr -> object_being_tracked = NO_OBJECT;
    }
    else
        printf ("BreakLockHandler: No missile for SensorId %d\n", sensor_id);
}
static REAL missile_maverick_detectibility (sensor_id, object, mav_loc,
                                   mav_boresight,
                                   flags)
int sensor_id;
TObjectP object;
VECTOR mav_loc;
VECTOR mav_boresight;
int flags;
 {
     REAL detectibility;
     VECTOR target_location;
     VECTOR to_target;
     REAL dotProduct;
     MAVERICK_MISSILE *mvptr;
     /* Get location of object */
     GetLocationOfTObject (object, target_location);
```

```
/* Determine detectibility. This is the cosine squared of the angle
    * between a vector from the sensor to the object and the boresight of
    * the sensor (for now).
    */
   /* Some of these computations may be duplicated in the tracking package.
    * May provide object calls to get them if that is more efficient.
    */
   vec_sub (target_location, mav_loc, to_target);
   dotProduct = vec_dot_prod (mav_boresight, to_target);
   detectibility = sign (dotProduct) * dotProduct * dotProduct /
                   vec_dot_prod (to_target, to_target);
   /* if the object is outside the detection cone of the sensor,
    * return a detectibility of 0.
   if ((mvptr = missile_maverick_get_missile_from_sensor_id (sensor_id))
     != NULL)
   {
     switch (mvptr -> mptr.state)
       case MAVERICK_READY:
         maverick_cone_threshold = MAVERICK_LOCK_THRESHOLD;
       case MAVERICK_FLYING:
         maverick_cone_threshold = MAVERICK_HOLD_THRESHOLD;
       case MAVERICK_FREE:
       default:
         printf ("MaverickDetectibility: Maverick not READY or FLYING\n");
         maverick_cone_threshold = MAYERICK_LOCK_THRESHOLD;
         break;
     }
     if (detectibility < maverick_cone_threshold)
            detectibility = 0.0;
    }
    else
     printf ("MaverickDetectibility: no missile for sensorID %d\n",
            sensor_id);
    return (detectibility);
static void missile_maverick_object_update ()
 * MissileMaverickSetPrelaunchIntervisibility
```

}

}

```
* Called from command line switch processing code to set the intervisibility
 * interface to use and the way to init it.
void missile_maverick_set_prelaunch_intervisibility_mode (mode)
char *mode;
    if (strlen (mode) > STRING_LEN)
      printf ("missile_maverick_set_prelaunch__intervisibility: type string too
long\n");
      return;
    strcpy (prelaunch_intervis_method, mode);
}
 * MissileMaverickSetLaunchedIntervisibility
 * Called from command line switch processing code to set the intervisibility
 * interface to use and the way to init it.
void missile_maverick_set_launched_intervisibility_mode (mode)
char *mode;
    if (strlen (mode) > STRING_LEN)
      printf ("missile_maverick_set_launched__intervisibility: type string too
 long\n");
      return;
     strcpy (in_flight_intervis_method, mode);
 }
 is maverick_flying (sensor_id)
 register int sensor_id;
 {
     register int i;
     for (i = 0; i < num_mavericks; i++)</pre>
       if (maverick_array[i].sensor_id == sensor_id)
           if (maverick_array[i].mptr.state == MAVERICK_FLYING)
               return (TRUE);
           else
               return (FALSE);
       }
     return (FALSE);
 }
 static void (*sensor_uninit_func) ();
 void sensor_uninit_callback (sensor_id)
```

```
int sensor_id;
{
    (*sensor_uninit_func) ();
}

missile_maverick_prepare_to_uninit_seeker (mvptr, uninit_func)

MAVERICK_MISSILE *mvptr;

void (*uninit_func) ();
{
    sensor_uninit_func = uninit_func;
    TrackSensorUnInitPrep (mvptr -> sensor_id, sensor_uninit_callback);
}
```

# Appendix J - Source code listing for miss\_nlos.c.

The following appendix contains the source code listing for miss\_nlos.c for convenience in document maintenance and understanding of the CSU.

```
/* $Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/libmissile/RCS/miss_nlo
s.c, v 1.4 1993/01/28 23:22:08 cm-adst Exp $ */
* $Log: miss_nlos.c,v $
* Revision 1.4 1993/01/28 23:22:08 cm-adst
 * P.DesMeules changes for spcr 31
* Revision 1.3 1993/01/06 21:13:50 cm-adst
 * R.Branson's changes for the weapson model.
 * Revision 1.1 1992/09/30 16:39:52 cm-adst
 * Initial Version
*/
static char RCS_ID[] = "$Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/li
bmissile/RCS/miss_nlos.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $";
/*****************************
 Revisions:
                                                         SP/CR Number
                        Author Title
                 Date
     Version
                 10/23/92 R. Branson Data File Initiali-
    1.2
                                      zation
                 10/30/92 R. Branson Added pathname to data
    1.3
                                      directory
                 11/25/92 R. Branson Changed %i to %d
    1.4
                 01/19/93 P.Desmeules Increased the size of the
     1.5
                                      fgets to make sure the
                                      whole line is read in.
    *************************
                    Description of Modification
       SP/CR No.
                     Hard coded defines changed to array elements.
                     Characteristics/parameter data array added.
                     Degree of polynomial data array added.
                     Added file reads for NLOS characteristics/
                        parameters, burn speed coefficients, and coast
                        speed coefficients.
                     Added "/simnet/data/" to each data file pathname.
       ******************************
 /******************************
            miss_nlos.c
  * FILE:
             Bryant Collard
  * AUTHOR:
```

```
* MAINTAINER: Bryant Collard
              This file contains routines which fly out a
 * PURPOSE:
              missile with the characteristics of a NLOS
              missile.
              11/25/88 bryant: Creation
 * HISTORY:
               4/24/89 bryant: Added static memory allocation
              05/17/89 dan: changed hellfire to nlos
 * Copyright (c) 1988 BBN Systems and Technologies, Inc.
 * All rights reserved.
  **********************************
#include "stdio.h"
#include "math.h"
#include "sim_types.h"
#include "sim_dfns.h"
#include "mass_stdc.h"
#include *dgi_stdg.h"
#include *sim_cig_if.h"
#include *protocol/pro_hdr.h"
#include "protocol/ammo.h"
#include "libmatrix.h"
#include "libmath.h"
#include "librva_util.h"
#include "libnear.h"
#include "miss_nlos.h"
#include "libmiss_dfn.h"
#include "libmiss_loc.h"
* Debug macro
#ifdef FILEDBG
#define P(a)
 #else
#define P(a)
 #endif
 /*/
 * Define missile characteristics.
                                     nlos_miss_char[ 0]
 #define NLOS_LOCK_THRESHOLD
                                     nlos_miss_char[ 1]
 #define NLOS_MAX_TURN_ANGLE
                                     nlos_miss_char[ 2]
 #define NLOS_VERTICAL_FLIGHT_TIME
                                      nlos_miss_char[ 3]
 #define NLOS_DECLINE_FLIGHT_TIME
                                           nlos_miss_char[ 4]
 #define NLOS_LEVEL_FLIGHT_TIME
                                      nlos_miss_char[5]
 #define NLOS_ARM_TIME
                                      nlos_miss_char[ 6]
 #define NLOS_BURNOUT_TIME
                                      nlos_miss_char[ 7]
 #define NLOS_MAX_FLIGHT_TIME
                                      nlos_miss_char[ 8]
 #define SPEED_0
```

```
nlos_miss_char[ 9]
#define SPEED_1
                     0.046542113 */ /*0.013962634*/
/*#define THETA_0
                                  nlos_miss_char[10]
#define THETA_0
* Set parameters which will control flight trajectory behavior.
                                  nlos_miss_char[11]
#define SIN_UNGUIDE
                                 nlos_miss_char[12]
#define COS_UNGUIDE
                           nlos_miss_char[13]
#define SIN_CLIMB
                           nlos_miss_char[14]
#define COS_CLIMB
                           nlos_miss_char[15]
#define SIN_LOCK
                           nlos_miss_char[16]
#define COS_LOCK
                           nlos_miss_char[17]
#define COS_TERM
                            nlos_miss_char[18]
#define COS_LOSE
 * The following terms set the order of the polynomials used to determine
 * the speed or cosine of the maximum allowed turn rate of the missile
 * at any point in time.
                             nlos_miss_poly_deg[0]
#define NLOS_BURN_SPEED_DEG
#define NLOS_COAST_SPEED_DEG nlos_miss_poly_deg[1]
/*/
 * NLOS missile characteristic parameters initialized to default values.
static REAL nlos_miss_char[20] =
    0.953153895, /* NLOS_LOCK_THRESHOLD */
                                                  radians/tick */
                    /* NLOS_MAX_TURN_ANGLE
    0.03490659,
                 /* NLOS_VERTICAL_FLIGHT_TIME */
   48.0.
                 /* NLOS_DECLINE_FLIGHT_TIME */
  105.0,
                /* NLOS_LEVEL_FLIGHT_TIME */
  140.0,
                                            ticks (1.3 sec) */
                /* NLOS_ARM_TIME
   20.0,
                                          ticks (1.5 sec) */
                /* NLOS_BURNOUT_TIME
   22.5,
                                            ticks (120 sec) */
                 /* NLOS_MAX_FLIGHT_TIME
  8000.0,
                       /* SPEED_0
   11.33333333,
    5.333333333, /* SPEED_1
                                      /*0.013962634*/
                  0.046542113 */
     /* THETA_0
    0.013962634, /* THETA_0 */
                                                deg */
     0.069756474, /* SIN_UNGUIDE
                                            4 deg */
     0.997564050, /* COS_UNGUIDE
                                            3.5 deg/sec */
     0.004072424, /* SIN_CLIMB
                                           3.5 deg/sec */
     0.999991708, /* COS_CLIMB
                                            9 deg */
     0.156434465, /* SIN_LOCK
                                           9 deg */
     0.987688341, /* COS_LOCK
                                           0 deg */
     0.984807753, /* COS_TERM
0.939692621, /* COS_LOSE
                                          20 deg */
     0.0
 };
 /*/
```

```
* The following terms set the order of the polynomials used to determine
* the speed and turn of the missile at any point in time.
/*/
static int nlos_miss_poly_deg[5] =
            /* Speed before motor burnout. */
  1,
            /* Speed after motor burnout. */
   3,
   0,
   0,
   0
};
* Coefficients for the speed polynomial before motor burnout.
/*/
static REAL nlos_burn_speed_coeff[5] =
{
                         /* a_0 - m/tick ( 67.0 m/sec)
    0.03333333,
                         /* a_1 - m/tick**2 (274.9732662 m/sec**2) */
    1.25777777,
    0.0,
    0.0,
    0.0
};
 * Coefficients for the speed polynomial after motor burnout.
static REAL nlos_coast_speed_coeff[5] =
                         /* a_0 - m/tick (327.2858074 m/sec) */
/* a_1 - m/tick**2 (-21.4609544 m/sec**2) */
    30.46972849,
    -9.7721160e-2,
                         /* a_2 - m/tick**3 ( 0.8227650 m/sec**3) */
     1.2433925e-4,
                         /* a_3 - m/tick**4 ( -0.0133200 m/sec**4) */
     -5.4061501e-8,
     0.0
};
static VECTOR nlos_initial_pos;
static VECTOR nlos_final_pos;
static VECTOR peak_target;
 static VECTOR decline_target;
 static VECTOR level_target;
 static int nlos_target_id;
 static int nlos_req_id;
 /*/
  * Declare static functions.
 static void missile_nlos_stop ();
```

```
* ROUTINE: missile_nlos_init
                 mptr - a pointer to the NLOS to be
  PARAMETERS:
                   initialized.
 * RETURNS: none
  PURPOSE: This routine initializes the state of the
           missile to indicate that it is available and
            sets values that never change.
void missile_nlos_init (mptr)
MISSILE *mptr;
{
      int
            data_tmp_int;
      int
      float data_tmp;
      char descript[80];
      FILE *fp;
      P(printf("$$$$ NLOS missile file data $$$$\n"););
/* DEFAULT CHARACTERISTICS DATA FOR miss_nlos.c READ FROM FILE
      fp = fopen("/simnet/data/ms_nl_ch.d", "r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_nl_ch.d\n");
            exit();
      }
      rewind(fp);
                             */
            Read array data
      i=0;
      while(fscanf(fp, "%f", &data_tmp) != EOF){
             nlos_miss_char[i] = data_tmp;
             fgets(descript, 80, fp);
             P(printf("nlos_miss_char(%3d) is%11.3f %s", i,
             nlos_miss_char[i], descript););
             ++i;
       }
       fclose(fp);
    END DEFAULT CHARACTERISTICS DATA FOR miss_nlos.c READ FROM FILE
                                                                    */
    DEFAULT BURN SPEED DATA FOR miss_nlos.c READ FROM FILE
       fp = fopen("/simnet/data/ms_nl_bs.d","r");
       if(fp==NULL){
             fprintf(stderr, "Cannot open /simnet/data/ms_nl_bs.d\n");
             exit();
       rewind(fp);
             Read degree of polynomial */
```

```
fscanf(fp, "%d", &data_tmp_int);
    NLOS_BURN_SPEED_DEG = data_tmp_int;
    fgets(descript, 80, fp);
    P(printf("nlos_miss_poly_deg(0) is%3d %s", NLOS_BURN_SPEED_DEG,
              descript););
          Read array data
    /*
    i=0;
    while(fscanf(fp, "%f", &data_tmp) != EOF){
          nlos_burn_speed_coeff[i] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("nlos_burn_speed_coeff(%3d) is%11.3f %s", i,
                       nlos_burn_speed_coeff[i], descript););
          ++i;
    }
    fclose(fp);
  END DEFAULT BURN SPEED DATA FOR miss_nlos.c READ FROM FILE
                                                                  */
  DEFAULT COAST SPEED DATA FOR miss_nlos.c READ FROM FILE
                                                                  */
     fp = fopen("/simnet/data/ms_nl_cs.d","r");
     if(fp==NULL){
           fprintf(stderr, "Cannot open /simnet/data/ms_nl_cs.d\n");
           exit();
     }
     rewind(fp);
           Read degree of polynomial */
     fscanf(fp, "%d", &data_tmp_int);
     NLOS_COAST_SPEED_DEG = data_tmp_int;
     fgets(descript, 80, fp);
     P(printf("nlos_miss_poly_deg(1) is%3d %s", NLOS_COAST_SPEED_DEG,
               descript););
                              */
     /*
           Read array data
     i=0;
     while(fscanf(fp, "%f", &data_tmp) != EOF){
           nlos_coast_speed_coeff[i] = data_tmp;
            fgets(descript, 80, fp);
           P(printf("nlos_coast_speed_coeff(%3d) is%11.3f %s", i,
                        nlos_coast_speed_coeff[i], descript););
            ++i;
     }
      fclose(fp);
/* END DEFAULT COAST SPEED DATA FOR miss_nlos.c READ FROM FILE
  mptr->state = FALSE;
    mptr->max_flight_time = NLOS_MAX_FLIGHT_TIME;
    mptr->max_turn_directions = 1;
    mptr->speed = SPEED_0;
```

```
mptr->cos_max_turn[0] = cos (NLOS_MAX_TURN_ANGLE);
   nlos_req_id = NEAR_NO_REQUEST_PENDING;
   nlos_target_id = vehicleIDIrrelevant;
}
               ***********
  ROUTINE: missile_nlos_fire
                 mptr - A pointer to the NLOS missile that *
 * PARAMETERS:
                  is to be launched.
           launch_point - The location in world
                         coordinates that the missile is
                         launched from.
           launch_to_world - The transformation matrix of
                            the launch platform to the
                            world.
           launch_speed - The speed of the launch
                         platform (assumed to be in the
                         direction of the missile). *
           tube - The tube the missile was launched from.
 * RETURNS: none
 * PURPOSE: This routine performs the functions
           specifically related to the firing of a
           Hellfire missile.
           *****************
void missile_nlos_fire (mptr, launch_point, launch_to_world, launch_speed,
       tube)
MISSILE *mptr;
VECTOR launch_point;
T_MATRIX launch_to_world;
REAL launch_speed;
int tube;
{
/*/
 * Set the initial time, location, orientation, and speed of the generic
   missile.
1*1
   mptr->time = 0.0;
   mptr->speed = SPEED_0;
    vec_copy (launch_point, mptr->location);
    vec_copy (launch_point, nlos_initial_pos);
   mat_copy (launch_to_world, mptr->orientation);
   mptr->init_speed = launch_speed;
  Tell the rest of the world about the firing of the missile. If this
   cannot be done, return.
/*/
    if (!missile_util_comm_fire_missile (mptr, MSL_TYPE_MISSILE,
                                        ammoHellfire, EFF_HELLFIRE,
                                        vehicleIDIrrelevant, targetUnknown,
                                        fuzePointDetonating, tube))
        return;
```

```
* If all was successful, set the missile state to TRUE and return.
  mptr->state = TRUE;
   peak_target[X] = 0.0;
   peak_target[Y] = 1000.0;
   peak_target[Z] = 1000.0;
   vec_mat_mul (peak_target, mptr->orientation, peak_target);
   vec_add (mptr->location, peak_target, peak_target);
   printf("peak_target: x = %f, y = %f, z = %f\n",
     peak_target[X],
     peak_target[Y],
     peak_target[Z]);
   decline_target[X] = 0.0;
   decline_target[Y] = 1800.0;
   decline_target[Z] = 0.0;
   vec_mat_mul (decline_target, mptr->orientation, decline_target);
   vec_add (mptr->location, decline_target, decline_target);
   printf("decline_target: x = %f, y = %f, z = %f n",
     decline_target[X],
     decline_target[Y],
     decline_target[Z]);
   level_target[X] = 0.0;
   level_target[Y] = 2000.0;
   level_target[Z] = 300.0;
   vec_mat_mul (level_target, mptr->orientation, level_target);
   vec_add (mptr->location, level_target, level_target);
   printf("level_target: x = %f, y = %f, z = %f\n",
     level_target[X],
     level_target[Y],
     level_target[Z]);
   return;
  ROUTINE: missile_nlos_fly
                 mptr - A pointer to the NLOS missile that *
 * PARAMETERS:
                   is to be flown out.
            target_location - The location in world
                              coordinates of the target.
 * RETURNS: none
 * PURPOSE: This routine performs the functions
            specifically related to the flying a NLOS *
            missile.
void missile_nlos_fly (mptr, nlos_target_loc, target_scheme)
MISSILE *mptr;
VECTOR nlos_target_loc;
```

```
int target_scheme;
                                   /* The current time after launch (ticks). */
    register REAL time;
    register REAL temp;
                                    /* A pointer to the target vehicles
    VehicleAppearancePDU *target;
                                   appearance packet. */
/*
    timed_printf("target_scheme = %d\nloc %f %f %f\n",
      target_scheme,
      nlos_target_loc[0],
      nlos_target_loc[1],
      nlos_target_loc[2]
*/
   Set and _time_. This is created mostly for increased readablity.
/*/
    time = mptr->time;
    if (time > 800.0)
      mptr->speed = SPEED_1;
 * choose the correct targettting option depending on flight time
/*/
if (time == NLOS_LEVEL_FLIGHT_TIME)
    printf("extra_waypoint: %f %f %f\n",
      mptr->location[0],
      mptr->location[1],
      mptr->location[2]);
    if (time < NLOS_VERTICAL_FLIGHT_TIME)
      missile_nlos_fly_to_point(mptr, peak_target);
    else if (time < NLOS_DECLINE_FLIGHT_TIME)</pre>
      missile_nlos_fly_to_point(mptr, decline_target);
    else if (time < NLOS_LEVEL_FLIGHT_TIME)</pre>
       level_target[Z] = mptr->location[Z];
      missile_nlos_fly_to_point(mptr, level_target);
    else
       switch (target_scheme)
           case NLOS_FLY_TO_POINT_IN_SPACE:
             missile_nlos_fly_to_point(mptr, nlos_target_loc);
             break;
           case NLOS_FLY_TO_POINT_RELATIVE:
             missile_target_nlos(mptr, nlos_target_loc);
             break;
           case NLOS_FLY_TO_TARGET:
             target = near_get_preferred_veh_near_vector (
```

```
&nlos_target_id,
                                    RVA_ALL_VEH,
                              mptr->location,
                              mptr->orientation[1],
                                    NLOS_LOCK_THRESHOLD,
                              &nlos_req_id);
            if (target != NULL)
                {
                timed_printf("miss_nlos: target locked on\n");
                missile_target_pursuit (mptr, target);
            else
                missile_target_unguided(mptr);
           break;
          default:
           printf("missile_nlos_fly: bad target_scheme\n");
          }
     }
 * check to see if the missile is "out of gas"
/*/
    if (mptr->time > 1500.0)
     mptr->target[Z] = 0.0;
   Try to actually fly the missile. If this fails stop the missile altogether
   and return.
/*/
    if (!missile_util_flyout (mptr))
       missile_nlos_stop (mptr);
      if (target_scheme == NLOS_FLY_TO_TARGET)
          nlos_target_id = vehicleIDIrrelevant;
          nlos_req_id = NEAR_NO_REQUEST_PENDING;
        return;
    }
    else
    {
/*/
        If the missile successfully flew, check for an intersection with the
        ground or a vehicle. If one is found, blow up the missile, stop its
        flyout and return.
/*/
        if (missile_util_comm_check_intersection (mptr, MSL_TYPE_MISSILE))
            missile_util_comm_check_detonate (mptr, MSL_TYPE_MISSILE);
            missile_nlos_stop (mptr);
            return;
```

```
}
/*/
* If the missile is to continue to fly, return.
/*/
   return;
 * ROUTINE: missile_nlos_stop
 * PARAMETERS: mptr - A pointer to the NLOS missile that *
                 is to be stopped.
 * RETURNS: none
 * PURPOSE: This routine causes all concerned to forget
           about the missile. It should be called when
            the flyout of any NLOS missile is stopped *
            (whether or not it has exploded). Note that
            this routine can only be called within this
            module.
static void missile_nlos_stop (mptr)
MISSILE *mptr;
 * Tell the world to stop worrying about this missile then release the
 * memory for use by other missiles.
    printf("initial_pos = %f %f %f\n",
            nlos_initial_pos[0],
            nlos_initial_pos[1],
            nlos_initial_pos[2]);
    printf("final_position = %f %f %f\n",
            mptr->location[0],
            mptr->location[1],
            mptr->location[2]);
    missile_util_comm_stop_missile (mptr, MSL_TYPE_MISSILE);
    mptr->state = FALSE;
```

## Appendix K - Source code listing for miss\_stinger.c.

The following appendix contains the source code listing for miss\_stinger.c for convenience in document maintenance and understanding of the CSU.

```
/* $Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/libmissile/RCS/miss_sti
nger.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $ */
 * $Log: miss_stinger.c,v $
 * Revision 1.4 1993/01/28 23:22:08 cm-adst
 * P.DesMeules changes for spcr 31
 * Revision 1.1 1992/09/30 16:39:52 cm-adst
 * Initial Version
*/
static char RCS_ID[] = "$Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/li
bmissile/RCS/miss_stinger.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $";
   **************
* Revisions:
                                                        SP/CR Number
                                   Title
                         Author
     Version
                 Date
                 10/23/92 R. Branson Data File Initiali-
     1.2
                                     zation
                 10/30/92 R. Branson Added pathname to data
    1.3
                                     directory
                 11/25/92 R. Branson Changed %i to %d
     1.4
                 01/12/93 P.J.Desmeules Increased the size
     1.5
                                     of the fgets to make
                                     sure the whole line is
                                     read.
      ******************************
       *************************
                     Description of Modification
       SP/CR No.
                     Hard coded defines changed to array elements.
                     Characteristics/parameter data array added.
                     Degree Of polynomial data array added.
                     Added file reads for stinger characteristics/
                       parameters, burn speed coefficients, and coast
                        speed coefficients.
                     Added "/simnet/data/" to each data file pathname.
       *******************************
 /***********************
             miss_stinger.c
  * FILE:
             Bryant Collard
  * AUTHOR:
  * MAINTAINER: Bryant Collard
```

```
This file contains routines which fly out a
* PURPOSE:
             missile with the characteristics of a STINGER
             missile.
             12/08/88 bryant: Creation
  HISTORY:
             04/24/89 bryant: Added static memory allocation
             08/07/90 bryant: NIU librva modifications.
* Copyright (c) 1988 BBN Systems and Technologies, Inc.
* All rights reserved.
*************************************
#include "stdio.h"
#include "math.h"
#include "sim_types.h"
#include "sim_dfns.h"
#include "basic.h"
#include "mun_type.h"
#include "libmap.h"
#include "libmatrix.h"
#include "libnear.h"
/*-- need Range_Squared info --*/
#include "libhull.h"
#include "libkin.h"
/*----*/
#include "miss_stinger.h"
#include "libmissile.h"
#include "libmiss_dfn.h"
#include "libmiss_loc.h"
* Debug macro
*/
#ifdef FILEDBG
#define P(a)
#else
#define P(a)
#endif
* Define missile characteristics.
/*/
                               stinger_miss_char[ 0]
#define STINGER_BURNOUT_TIME
#define STINGER_MAX_FLIGHT_TIME stinger_miss_char[ 1]
#define STINGER_LOCK_THRESHOLD stinger_miss_char[ 2]
                                stinger_miss_char[ 3]
#define SPEED_0
                               stinger_miss_char[ 4]
#define THETA_0
                               stinger_miss_char[ 5]
#define INVEST_DIST_SQ
                               stinger_miss_char[ 6]
#define FUZE_DIST_SQ
/*/
```

```
* Define the states the _STINGER_MISSILE_ can be in.
/*/
                         /* No missile assigned. */
/* Missile assigned to ready state. */
#define STINGER_FREE 0
#define STINGER_READY 1
                           /* Missile assigned to flying state. */
#define STINGER_FLYING 2
/*/
 * The following terms set the order of the polynomials used to determine
 * the speed of the missile at any point in time.
static int stinger_miss_poly_deg[2] =
           /* burn speed poly degree */
      /* coast speed poly degree */
};
/*/
* Stinger missile characteristic parameters initialized to default values.
static REAL stinger_miss_char[15] =
                  /* ticks (1.275 sec) */
    19.125,
                  /* ticks (26.667 sec) */
   400.000,
                      /* cos (12.5 deg) ** 2 */
     0.953153895,
                        /* m/tick (800 m/sec) */
    53.3333333,
                        /* rad/tick (15.0 deg/sec) */
     0.0174,
                        /* (300 m) ** 2 */
 90000.0,
                        /* (20 m) ** 2 */
   400.0,
     0.0,
     0.0,
     0.0,
     0.0,
     0.0,
     0.0,
     0.0,
     0.0
};
 /*/
 * Coefficients for the speed polynomial before motor burnout initialized to
 * default values.
 /*/
 static REAL stinger_burn_speed_coeff[STINGER_BURN_SPEED_DEG + 1] =
 {
                          /* a_0 - m/tick */
     1.9.
                          /* a_1 - m/tick**2 */
     2.689324619
 };
 /*/
 * Coefficients for the speed polynomial after motor burnout initialized to
 * default values.
 /*/
```

```
static REAL stinger_coast_speed_coeff[STINGER_COAST_SPEED_DEG + 1] =
   };
/*/
 * Memory for the missiles is declared in vehicle specific code. During
 * initialization, a pointer is assigned to this memory then all memory
* issues are dealt with in this module.
/*/
static STINGER_MISSILE *stinger_array; /* A pointer to missile memory. */
                                    /* The number of defined missiles. */
static int num_stingers;
static ObjectType stinger_ammo_type = munition_US_Stinger;
static REAL
max_range_limit, /* [ MISSILE_US_MAX_RANGE_LIMIT ]
max_range_squared, /* [ MISSILE_US_MAX_RANGE_LIMIT ^ 2 ]
speed_factor; /* [ MISSILE_US_SPEED_FACTOR ]
                                                                     */
/*/
 * Declare static functions.
/*/
static void missile_stinger_fly ();
/********************
 * ROUTINE: missile_stinger_init
 * PARAMETERS: missile_array - A pointer to an array of *
                         STINGER missiles defined in
                          vehicle specific code.
           num_missiles - The number missiles defined in
                         _missile_array_.
 * RETURNS: none
 * PURPOSE: This routine copies the parameters into
            variables static to this module and initializes *
            the state of all the missiles. It also
            initializes the proximity fuze.
 *************************************
void missile_stinger_init (missile_array, num_missiles)
 STINGER_MISSILE missile_array[];
 int num_missiles;
 {
             /* A counter. */
    int i;
            int j;
      int data_tmp_int;
      float data_tmp;
      char descript[80];
```

```
FILE *fp;
   P(printf("$$$$$ STINGER missile file data $$$$\n"););
DEFAULT CHARACTERISTIC DATA FOR miss_stinger.c READ FROM FILE */
   fp = fopen("/simnet/data/ms_st_ch.d", "r");
   if(fp==NULL){
         fprintf(stderr, "Cannot open /simnet/data/ms_st_ch.d\n");
         exit();
   }
   rewind(fp);
         Read array data
   j=0;
   while(fscanf(fp, "%f", &data_tmp) != EOF){
         stinger_miss_char[j] = data_tmp;
         fgets(descript, 80, fp);
         P(printf("stinger_miss_char(%3d) is%11.3f %s", j,
               stinger_miss_char[j],
                     descript););
         ++j;
   }
   fclose(fp);
 END DEFAULT CHARACTERISTIC DATA FOR miss_stinger.c READ FROM FILE
 DEFAULT BURN SPEED DATA FOR miss_stinger.c READ FROM FILE
                                                                 */
   fp = fopen("/simnet/data/ms_st_bs.d", "r");
   if(fp==NULL){
         fprintf(stderr, "Cannot open /simnet/data/ms_st_bs.d\n");
         exit();
   rewind(fp);
                                        */
         Read degree of polynomial
   fscanf(fp, "%d", &data_tmp_int);
   stinger_miss_poly_deg[0] = data_tmp_int;
   fgets(descript, 80, fp);
   P(printf("stinger_miss_poly_deg(0) is%3d %s",
   stinger_miss_poly_deg[0], descript););
         Read array data
                            */
   /*
   j=0;
   while(fscanf(fp, "%f", &data_tmp) != EOF) {
          stinger_burn_speed_coeff[j] = data_tmp;
          fgets(descript, 80, fp);
          P(printf("stinger_burn_speed_coeff(%3d) is%11.3f %s", j,
                stinger_burn_speed_coeff[j],
                      descript););
          ++j;
```

```
}
     fclose(fp);
   END DEFAULT BURN SPEED DATA FOR miss_stinger.c READ FROM FILE */
  DEFAULT COAST SPEED DATA FOR miss_stinger.c READ FROM FILE
     fp = fopen("/simnet/data/ms_st_cs.d","r");
     if(fp==NULL){
           fprintf(stderr, "Cannot open /simnet/data/ms_st_cs.d\n");
           exit();
     }
     rewind(fp);
                                          */
           Read degree of polynomial
     fscanf(fp, "%d", &data_tmp_int);
     stinger_miss_poly_deg[1] = data_tmp_int;
     fgets(descript, 80, fp);
     P(printf("stinger_miss_poly_deg(1) is%3d %s",
           stinger_miss_poly_deg[1], descript););
           Read array data
                            */
     j=0;
     while(fscanf(fp, "%f", &data_tmp) != EOF){
            stinger_coast_speed_coeff[j] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("stinger_coast_speed_coeff(%3d) is%11.3f %s", j,
                  stinger_coast_speed_coeff[j],
                        descript););
            ++j;
      }
      fclose(fp);
                                                                         */
/* END DEFAULT COAST SPEED DATA FOR miss_stinger.c READ FROM FILE
   num_stingers = num_missiles;
    stinger_array = missile_array;
    for (i = 0; i < num_missiles; i++)
        stinger_array[i].mptr.state = STINGER_FREE;
        stinger_array[i].mptr.max_flight_time = STINGER_MAX_FLIGHT_TIME;
        stinger_array[i].mptr.max_turn_directions = 1;
    speed_factor = MISSILE_US_SPEED_FACTOR;
    max_range_limit = MISSILE_US_MAX_RANGE_LIMIT;
    max_range_squared = max_range_limit * max_range_limit;
    stinger_ammo_type = munition_US_Stinger;
    Initialize the proximity fuze.
/*/
    missile_fuze_prox_init ();
```

```
void missile_stinger_set_speed_factor( scale_speed )
REAL scale_speed;
    speed_factor = scale_speed;
}
void missile_stinger_set_max_range_limit( limit_range )
REAL limit_range;
{
   max_range_limit = limit_range;
   max_range_squared = max_range_limit * max_range_limit;
}
void missile_stinger_set_ammo_type( ammo )
ObjectType ammo;
    stinger_ammo_type = ammo;
}
 * ROUTINE: missile_stinger_ready
 * PARAMETERS:
                none
 * RETURNS: A pointer to a missile that is currently
            available.
 * PURPOSE: This routine finds, if possible, a missile that *
            is not being used, puts it in a ready state and *
            returns a pointer to it.
 ************************
STINGER_MISSILE *missile_stinger_ready ()
             /* A counter. */
    int i;
/*/
    Try to find a free missile.
    for (i = 0; i < num_stingers; i++)</pre>
/*/
        If a free missile is found, put it in a ready state, clear the target
        ID and return a pointer to it.
/*/
        if (stinger_array[i].mptr.state == STINGER_FREE)
            stinger_array[i].mptr.state = STINGER_READY;
            stinger_array[i].target_vehicle_id.vehicle = vehicleIrrelevant;
            return (&stinger_array[i]);
        }
    }
    If no free missile is found, return a NULL pointer.
```

```
return (NULL);
}
              *************
  ROUTINE: missile_stinger_pre_launch
                sptr - A pointer to the missile that is to be
  PARAMETERS:
                 serviced.
           launch_point - The location of the missile in
                        world coordinates.
           launch_to_world - The transformation matrix of
                           the missile to the world.
              veh_list - Vehicle list ID.
 * RETURNS: none
 * PURPOSE: This routine is called after a missile has been *
           readied and before it has been launched. It
           determines if the seeker head can see a target *
           and, if it can see a target, stores its
           position.
   **********
void missile_stinger_pre_launch (sptr, launch_point, launch_to_world, veh_list)
STINGER_MISSILE *sptr;
VECTOR launch_point;
T_MATRIX launch_to_world;
int veh_list;
{
   VehicleAppearanceVariant *target; /* A pointer to the target vehicles
                                       appearance packet. */
 * Try to find a target.
/*/
   target = near_get_preferred_veh_near_vector (&(sptr->target_vehicle_id),
           veh_list, launch_point, launch_to_world[1],
           STINGER_LOCK_THRESHOLD);
 * If a target is found, store its location.
    if (target != NULL)
       sptr->target_vehicle_id = target->vehicleID;
       missile_target_pursuit (&(sptr->mptr), target->location);
    else
       sptr->target_vehicle_id.vehicle = vehicleIrrelevant;
}
        ****************
 * ROUTINE: missile_stinger_fire
 * PARAMETERS: sptr - A pointer to the STINGER missile that
                 is to be launched.
```

```
launch_point - The location in world
                          coordinates that the missile is
                          launched from.
           launch_to_world - The transformation matrix of
                             the launch platform to the
                             world.
           launch_speed - The speed of the launch
                          platform (assumed to be in the
                          direction of the missile). *
           tube - The tube the missile was launched from.
 * RETURNS: TRUE for a successful launch and FALSE for an
           unsuccessful launch.
  PURPOSE: This routine performs the functions
           specifically related to the firing of a
           STINGER missile.
 *********
int missile_stinger_fire (sptr, launch_point, launch_to_world, launch_speed,
        tube)
STINGER_MISSILE *sptr;
VECTOR launch_point;
T_MATRIX launch_to_world;
REAL launch_speed;
int tube;
                          /* Counter. */
    int i;
                           /* Pointer to the particular generic missile
    MISSILE *mptr;
                              pointed at by _sptr_. */
/*/
    Get a pointer to the generic elements of the STINGER missile. This
    improves code readability.
    mptr = &(sptr->mptr);
/*/
   Set the initial time, location, orientation and speed of the generic
   missile.
/*/
    mptr->time = 0.0;
    vec_copy (launch_point, mptr->location);
    mat_copy (launch_to_world, mptr->orientation);
    mptr->speed = launch_speed +
        (speed_factor *
         missile_util_eval_poly (STINGER_BURN_SPEED_DEG,
                                 stinger_burn_speed_coeff, 0.0));
    mptr->init_speed = launch_speed;
/*/
    Indicate that the proximity fuze has no vehicles it is tracking.
/*/
    sptr->pptr = NULL;
/*/
    Determine range equations for intercept targeting.
 /*/
    sptr->stinger_burn_range_coeff[0] = 0.0;
    for (i = 1; i <= STINGER_BURN_SPEED_DEG + 1; i++);</pre>
```

```
{
       sptr->stinger_burn_range_coeff[i] = (1.0 / ((REAL) i)) *
               stinger_burn_speed_coeff[i - 1];
   sptr->stinger_burn_range_coeff[1] += launch_speed;
   missile_target_intercept_find_poly (STINGER_COAST_SPEED_DEG, launch_speed,
           stinger_coast_speed_coeff, sptr->stinger_coast_range_coeff,
           sptr->stinger_coast_range_2_coeff);
/*/
 * Tell the rest of the world about the firing of the missile. If this
  cannot be done, release the missile memory and return FALSE.
   if (!missile_util_comm_fire_missile (mptr, MSL_TYPE_MISSILE,
           map_get_ammo_entry_from_network_type (stinger_ammo_type),
           stinger_ammo_type, stinger_ammo_type,
           &(sptr->target_vehicle_id), targetIsVehicle, objectIrrelevant,
           tube))
    {
       mptr->state = STINGER_FREE;
       return (FALSE);
    }
/** /
 * If all was successful, set the missile state to STINGER_FLYING and
 * return TRUE.
    mptr->state = STINGER_FLYING;
    return (TRUE);
}.
/************************
 * ROUTINE: missile_stinger_fly_missiles
 * PARAMETERS: veh_list - Vehicle list ID.
 * RETURNS: none
 * PURPOSE: This routine flies out all missiles in a *
            flying state.
 ************
void missile_stinger_fly_missiles (veh_list)
int veh_list;
    int i; /* A counter. */
 /*/
  * Fly out all flying missiles.
 /*/
    for (i = 0; i < num_stingers; i++)</pre>
        if (stinger_array[i].mptr.state == STINGER_FLYING)
            missile_stinger_fly (&(stinger_array[i]), veh_list);
     }
 }
```

```
* ROUTINE: missile_stinger_fly
  PARAMETERS: sptr - A pointer to the STINGER missile that
                 is to be flown out.
               veh_list - Vehicle list ID.
* RETURNS: none
* PURPOSE: This routine performs the functions
           specifically related to the flying a STINGER
           missile.
***********************************
static void missile_stinger_fly (sptr, veh_list)
STINGER_MISSILE *sptr;
int veh_list;
   register MISSILE *mptr; /* A pointer to the generic aspects of
                                     _sptr_. */
                                  /* The current time after launch (ticks). */
    REAL time;
   VehicleAppearanceVariant
                                  /* A pointer to the targets appearance
       *target;
                                     packet. */
/*/
 * Set _mptr_ and _time_. These values are created mostly for increased
 * readablity.
    mptr = &(sptr->mptr);
    time = mptr->time;
  Find the current missile speed and the cosine of the maximum allowed turn
    angle. The equations used are different before and after motor burnout.
/*/
    if (time < STINGER_BURNOUT_TIME)</pre>
        mptr->speed = missile_util_eval_poly (STINGER_BURN_SPEED_DEG,
                stinger_burn_speed_coeff, time) + mptr->init_speed;
    }
    else
        mptr->speed = missile_util_eval_poly (STINGER_COAST_SPEED_DEG,
                stinger_coast_speed_coeff, time) + mptr->init_speed;
    }
 /*/
 * Note that this is a temporary method of finding turn angle.
    mptr->cos_max_turn[0] = cos (sqrt (mptr->speed / (SPEED_0 +
            mptr->init_speed)) * THETA_0);
    Try to find a target. If one is found, fly towards it in the
    proper trajectory, otherwise, fly in a straight line.
 /*/
     target = near_get_preferred_veh_near_vector (&(sptr->target_vehicle_id),
            veh_list, mptr->location, mptr->orientation[1],
             STINGER_LOCK_THRESHOLD);
```

```
if( max_range_limit > 0 &&
     kinematics_range_squared (veh_kinematics, mptr->location) >
      max_range_squared )
       missile_target_ground( mptr );
   else if (target != NULL)
       sptr->target_vehicle_id = target->vehicleID;
       if (time < STINGER_BURNOUT_TIME)
           missile_target_intercept_pre_burnout (mptr, target,
                   sptr->stinger_burn_range_coeff, STINGER_BURNOUT_TIME,
                   STINGER_BURN_SPEED_DEG + 1,
                   sptr->stinger_coast_range_coeff,
                   sptr->stinger_coast_range_2_coeff,
                   STINGER_COAST_SPEED_DEG + 1);
       else
           missile_target_intercept (mptr, target,
                   sptr->stinger_coast_range_coeff,
                   sptr->stinger_coast_range_2_coeff,
                   STINGER_COAST_SPEED_DEG + 1);
   }
   else
       sptr->target_vehicle_id.vehicle = vehicleIrrelevant;
       missile_target_unguided (mptr);
   Try to actually fly the missile. If this fails, stop the missile
   altogether and return.
   if (!missile_util_flyout (mptr))
       missile_stinger_stop (sptr);
       return;
    }
   else
    {
/*/
        If the missile successfully flew, process the proximity fuze.
/*/
        if (sptr->target_vehicle_id.vehicle == vehicleIrrelevant)
            missile_fuze_prox (mptr, MSL_TYPE_MISSILE, PROX_FUZE_ON_ALL_VEH,
                    &(sptr->target_vehicle_id), &(sptr->pptr),
                    veh_list, INVEST_DIST_SQ, FUZE_DIST_SQ);
        else
            missile_fuze_prox (mptr, MSL_TYPE_MISSILE, PROX_FUZE_ON_ONE_VEH,
                    &(sptr->target_vehicle_id), &(sptr->pptr),
                    veh_list, INVEST_DIST_SQ, FUZE_DIST_SQ);
/*/
        If the missile has intersected of self detonated, blow it up, stop its
        flyout and return.
/*/
        if (missile_util_comm_check_detonate (mptr, MSL_TYPE_MISSILE))
            missile_stinger_stop (sptr);
            return;
```

```
}
* If the missile is to continue to fly, return.
   return;
 * ROUTINE: missile_stinger_stop
* PARAMETERS: sptr - A pointer to the STINGER missile that
                  is to be stopped.
 * RETURNS: none
 * PURPOSE: This routine causes all concerned to forget
           about the missile. It should be called when
           the flyout of any STINGER missile is stopped
            (whether or not it has exploded).
void missile_stinger_stop (sptr)
STINGER_MISSILE *sptr;
{
/*/
 * If the missile has been fired, tell the world to stop it and clear the
* proximity fuze targets. Release missile memory for use by other missiles.
    if (sptr->mptr.state == STINGER_FLYING)
        missile_util_comm_stop_missile (&(sptr->mptr), MSL_TYPE_MISSILE);
        missile_fuze_prox_stop (&(sptr->pptr));
    sptr->mptr.state = STINGER_FREE;
}
```

# Appendix L - Source code listing for miss\_tow.c.

The following appendix contains the source code listing for miss\_tow.c for convenience in document maintenance and understanding of the CSU.

```
/* SHeader: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/libmissile/RCS/miss_tow
.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $ */
* $Log: miss_tow.c,v $
* Revision 1.4 1993/01/28 23:22:08 cm-adst
 * P.DesMeules changes for spcr 31
 * Revision 1.3 1993/01/06 21:14:12 cm-adst
 * R.Branson's changes for the weapons model.
* Revision 1.1 1992/09/30 16:39:52 cm-adst
 * Initial Version
*/
static char RCS_ID[] = "$Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/li
bmissile/RCS/miss_tow.c,v 1.4 1993/01/28 23:22:08 cm-adst Exp $";
  ******************
 Revisions:
                                                          SP/CR Number
                          Author Title
     Version
                Date
                 10/23/92 R. Branson Data File Initiali-
     1.2
                                       zation
                10/30/92 R. Branson Added pathname to data
    1.3
                                       directory
                 11/25/92 R. Branson Changed %i to %d
     1.4
                 01/19/93 P.Desmeules Increased the size of the
     1.5
                                       fgets to make sure the
                                       whole line is read in.
/***************************
                     Description of Modification
       SP/CR No.
                     Hard coded defines changed to array elements.
                     Characteristics/parameter data array added.
                     Degree of polynomial data array added.
                     Added file reads for TOW characteristics/parameters,
                        burn speed coefficients, coast speed coefficients,
                        burn turn coefficients, and coast turn coeffi-
                        coefficients.
                     Added "/simnet/data/" to each data file pathname.
        ****************
 * FILE: miss_tow.c
```

```
Bryant Collard
* AUTHOR:
* MAINTAINER: Bryant Collard
             This file contains routines which fly out a
 PURPOSE:
             missile with the characteristics of a TOW
             missile.
             10/31/88 bryant: Creation
* HISTORY:
              4/26/89 bryant: Added statically allocated mem
* Copyright (c) 1988 BBN Systems and Technologies, Inc.
* All rights reserved.
 *************************
#include "stdio.h"
#include "sim_types.h"
#include "sim_dfns.h"
#include "basic.h"
#include "mun_type.h"
#include "libmatrix.h"
#include "libmap.h"
/*-- need Range_Squared info --*/
#include "libhull.h"
#include "libkin.h"
/*----*/
#include "miss_tow.h"
#include "libmissile.h"
#include "libmiss_dfn.h"
#include "libmiss_loc.h"
* Debug macro
*/
#ifdef FILEDBG
#define P(a)
#else
#define P(a)
#endif
 * Define missile characteristics.
#define TOW_BURNOUT_TIME
                            tow_miss_char[0]
#define TOW_RANGE_LIMIT_TIME tow_miss_char[1]
#define TOW_MAX_FLIGHT_TIME tow_miss_char[2]
/*/
    The following terms set the order of the polynomials used to determine
 * the speed or cosine of the maximum allowed turn rate of the missile
 * at any point in time.
/*/
```

```
#define TOW_BURN_SPEED_DEG tow_miss_poly_deg[0]
#define TOW_COAST_SPEED_DEG tow_miss_poly_deg[1]
#define TOW_BURN_TURN_DEG tow_miss_poly_deg[2]
#define TOW_COAST_TURN_DEG tow_miss_poly_deg[3]
/*/
* Tow missile characteristic parameters initialized to default values.
static REAL tow_miss_char[5] =
           /* ticks (1.6 sec) */
   24.0,
          /* ticks (17.89 sec) */
  268.35,
           /* ticks - cos of max turn > 1.0 beyond this point */
  300.00,
    0.0,
    0.0
};
  The following terms set the order of the polynomials used to determine
 * the speed and turn of the missile at any point in time.
/*/
static int tow_miss_poly_deg[5] =
            /* Speed before motor burnout. */
   2.
            /* Speed after motor burnout. */
   3,
            /* Cosine of max turn before burnout. */
   1,
           /* Cosine of max turn after burnout. */
   3,
            /* not used. */
};
/*/
 * Coefficients for the speed polynomial before motor burnout initialized
 * to default values.
/*/
static REAL tow_burn_speed_coeff[5] =
                         /* a_0 - m/tick (67.0 m/sec) */
    4.466666667,
                         /* a_1 - m/tick**2 (274.9732662 m/sec**2) */
    1.222103405,
                         /* a_2 - m/tick**3 (-82.7057910 m/sec**3) */
   -0.024532086,
    0.0,
    0.0
};
 * Coefficients for the speed polynomial after motor burnout.
static REAL tow_coast_speed_coeff[5] =
                          /* a_0 - m/tick (327.2858074 m/sec) */
    21.81905383,
                         /* a_1 - m/tick**2 (-21.4609544 m/sec**2) */
    -9.5382019e-2,
                         /* a_2 - m/tick**3 (0.8227650 m/sec**3) */
     2.4378222e-4,
                         /* a_3 - m/tick**4 (-0.0133200 m/sec**4) */
     -2.6311111e-7,
      0.0
```

```
};
/*/
 * Coefficients for the cosine of max turn polynomials before motor burnout.
 * The structure _MAX_COS_COEFF_ is used to store the values for the turn
 * sideways, up, and down polynomials along with their order.
static MAX_COS_COEFF tow_burn_turn_coeff =
                        /* Order of the polynomials. */
    1.
    {
                        /* Sidewards turn. */
                       /* a_0 - cos(rad)/tick */
        0.999976868652,
                        /* a_1 - cos(rad)/tick**2 */
       -3.5933955e-7
    },
    {
                        /* Upwards turn. */
        0.999960667258, /* a_0 - cos(rad)/tick */
                        /* a_1 - cos(rad)/tick**2 */
       -3.1492328e-6
    },
                         /* Downwards turn. */
        0.999978909989, /* a_0 - cos(rad)/tick */
                        /* a_1 - cos(rad)/tick**2 */
       -7.8194991e-9
    }
};
 * Coefficients for the cosine of max turn polynomials after motor burnout.
static MAX_COS_COEFF tow_coast_turn_coeff =
{
                         /* Order of the polynomials. */
    3,
    {
                         /* Sidewards turn. */
                        /* a_0 - cos(rad)/tick */
        0.99995112518,
                        /* a_1 - cos(rad)/tick**2 */
        8.96333e-7,
                        /* a_2 - cos(rad)/tick**3 */
       -5.995375e-9,
                        /* a_3 - cos(rad)/tick**4 */
        1.162225e-11
    },
     {
                          /* Upwards turn. */
                         /* a_0 - cos(rad)/tick */
        0.9998498495,
                         /* a_1 - cos(rad)/tick**2 */
        1.657779e-6,
                         /* a_2 - cos(rad)/tick**3 */
        -8.231861e-9,
                         /* a_3 - cos(rad)/tick**4 */
        1.381832e-11
     },
     {
                          /* Downwards turn. */
                         /* a_0 - cos(rad)/tick */
        0.9999714014,
                         /* a_1 - cos(rad)/tick**2 */
        3.382077e-7,
                         /* a_2 - cos(rad)/tick**3 */
        -1.601259e-9,
                         /* a_3 - cos(rad)/tick**4 */
        2.623014e-12
```

```
}
};
static ObjectType tow_ammo_type = munition_US_TOW;
static REAL
   max_range_limit, /* [ MISSILE_US_MAX_RANGE_LIMIT ]
                                                                      */
   max_range_squared, /* [ MISSILE_US_MAX_RANGE_LIMIT ^ 2 ]
                                                                      */
   speed_factor; /* [ MISSILE_US_SPEED_FACTOR ]
/*/
 * Declare static functions.
/*/
static void missile_tow_stop ();
 * ROUTINE: missile_tow_init
 * PARAMETERS: tptr - a pointer to the TOW to be
                  initialized.
 * RETURNS: none
 * PURPOSE: This routine initializes the state of the
           missile to indicate that it is available and
           sets values that never change.
      **********************************
void missile_tow_init (tptr)
TOW_MISSILE *tptr;
{
      int
          i;
           data_tmp_int;
      int
      float data_tmp;
      char descript[80];
      FILE *fp;
      P(printf("$$$$ TOW missile file data $$$$\n"););
/* DEFAULT CHARACTERISTICS DATA FOR miss_tow.c READ FROM FILE
      fp = fopen("/simnet/data/ms_tw_ch.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_tw_ch.d\n");
            exit();
       }
      rewind(fp);
            Read array data */
       /*
       i=0;
       while(fscanf(fp, "%f", &data_tmp) != EOF){
             tow_miss_char[i] = data_tmp;
             fgets(descript, 80, fp);
            P(printf("tow_miss_char(%3d) is%11.3f %s", i, tow_miss_char[i],
                        descript););
```

```
++i;
     }
     fclose(fp);
   END DEFAULT CHARACTERISTICS DATA FOR miss_tow.c READ FROM FILE
                                                                         */
                                                                   */
  DEFAULT BURN SPEED DATA FOR miss_tow.c READ FROM FILE
     fp = fopen("/simnet/data/ms_tw_bs.d","r");
     if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_tw_bs.d\n");
           exit();
     }
     rewind(fp);
           Read degree of polynomial */
     fscanf(fp, "%d", &data_tmp_int);
     TOW_BURN_SPEED_DEG = data_tmp_int;
     fgets(descript, 80, fp);
     P(printf("tow_miss_poly_deg(0) is%3d %s", TOW_BURN_SPEED_DEG,
                descript););
           Read array data
     i=0:
     while(fscanf(fp, "%f", &data_tmp) != EOF){
            tow_burn_speed_coeff[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("tow_burn_speed_coeff(%3d) is%11.3f %s", i,
                        tow_burn_speed_coeff[i], descript););
            ++i;
      }
      fclose(fp);
/* END DEFAULT BURN SPEED DATA FOR miss_tow.c READ FROM FILE
                                                                   */
/* DEFAULT COAST SPEED DATA FOR miss_tow.c READ FROM FILE
      fp = fopen("/simnet/data/ms_tw_cs.d", "r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/ms_tw_cs.d\n");
            exit();
      }
     rewind(fp);
            Read degree of polynomial */
      fscanf(fp, "%d", &data_tmp_int);
      TOW_COAST_SPEED_DEG = data_tmp_int;
      fgets(descript, 80, fp);
      P(printf("tow_miss_poly_deg(1) is%3d %s", TOW_COAST_SPEED_DEG,
                descript););
            Read array data
```

```
i=0;
  while(fscanf(fp, "%f", &data_tmp) != EOF){
        tow_coast_speed_coeff[i] = data_tmp;
        fgets(descript, 80, fp);
        P(printf("tow_coast_speed_coeff(%3d) is%11.3f %s", i,
                    tow_coast_speed_coeff[i], descript););
        ++i;
  }
  fclose(fp);
END DEFAULT COAST SPEED DATA FOR miss_tow.c READ FROM FILE
DEFAULT BURN TURN DATA FOR miss_tow.c READ FROM FILE
                                                                */
  fp = fopen("/simnet/data/ms_tw_bt.d","r");
  if(fp==NULL){
        fprintf(stderr, "Cannot open /simnet/data/ms_tw_bt.d\n");
        exit();
  }
  rewind(fp);
        Read degree of polynomial */
  fscanf(fp, "%d", &data_tmp_int);
  TOW BURN_TURN_DEG = data_tmp_int;
  tow_burn_turn_coeff.deg = data_tmp_int;
  fgets(descript, 80, fp);
  P(printf("tow_miss_poly_deg(2) is%3d %s", TOW_BURN_TURN_DEG,
             descript););
        Read array data
                           */
  /*
  for (i=0; i <= data_tmp_int; i++) {</pre>
         fscanf(fp, "%f", &data_tmp);
         tow_burn_turn_coeff.side_coeff[i] = data_tmp;
         fgets(descript, 80, fp);
         P(printf("tow_burn_turn_coeff.side_coeff(%3d) is%11.3f %s", i,
                     tow_burn_turn_coeff.side_coeff[i], descript););
   }
   for (i=0; i <= data_tmp_int; i++) {
         fscanf(fp, "%f", &data_tmp);
         tow_burn_turn_coeff.up_coeff[i] = data_tmp;
         fgets(descript, 80, fp);
         P(printf("tow_burn_turn_coeff.up_coeff(%3d) is%11.3f %s", i,
                     tow_burn_turn_coeff.up_coeff[i], descript););
   }
   for (i=0; i <= data_tmp_int; i++) {
         fscanf(fp, "%f", &data_tmp);
         tow_burn_turn_coeff.down_coeff[i] = data_tmp;
         fgets(descript, 80, fp);
         P(printf("tow_burn_turn_coeff.down_coeff(%3d) is%11.3f %s", i,
                      tow_burn_turn_coeff.down_coeff[i], descript););
```

```
}
    fclose(fp);
  END DEFAULT BURN TURN DATA FOR miss_tow.c READ FROM FILE
                                                                  */
  DEFAULT COAST TURN DATA FOR miss_tow.c READ FROM FILE
                                                                  */
    fp = fopen("/simnet/data/ms_tw_ct.d","r");
    if(fp==NULL){
          fprintf(stderr, "Cannot open /simnet/data/ms_tw_ct.d\n");
     }
    rewind(fp);
           Read degree of polynomial */
     fscanf(fp, "%d", &data_tmp_int);
    TOW_COAST_TURN_DEG = data_tmp_int;
     tow_coast_turn_coeff.deg = data_tmp_int;
     fgets(descript, 80, fp);
     P(printf("tow_miss_poly_deg(3) is%3d %s", TOW_COAST_TURN_DEG,
               descript););
           Read array data
     for (i=0; i <= data_tmp_int; i++) {</pre>
           fscanf(fp, "%f", &data_tmp);
           tow_coast_turn_coeff.side_coeff[i] = data_tmp;
           fgets(descript, 80, fp);
           P(printf("tow_coast_turn_coeff.side_coeff(%3d) is%11.3f %s", i,
                       tow_coast_turn_coeff.side_coeff[i], descript););
     }
     for (i=0; i <= data_tmp_int; i++) {
           fscanf(fp, "%f", &data_tmp);
           tow_coast_turn_coeff.up_coeff[i] = data_tmp;
           fgets(descript, 80, fp);
           P(printf("tow_coast_turn_coeff.up_coeff(%3d) is%11.3f %s", i,
                        tow_coast_turn_coeff.up_coeff[i], descript););
     }
     for (i=0; i <= data_tmp_int; i++) {</pre>
            fscanf(fp, "%f", &data_tmp);
            tow_coast_turn_coeff.down_coeff[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("tow_coast_turn_coeff.down_coeff(%3d) is%11.3f %s", i,
                        tow_coast_turn_coeff.down_coeff[i], descript););
    . }
      fclose(fp);
/* END DEFAULT COAST TURN DATA FOR miss_tow.c READ FROM FILE
                                                                    */
   tptr->mptr.state = FALSE;
   tptr->mptr.max_flight_time = TOW_MAX_FLIGHT_TIME;
    tptr->mptr.max_turn_directions = 3;
```

```
speed_factor = MISSILE_US_SPEED_FACTOR;
   max_range_limit = MISSILE_US_MAX_RANGE_LIMIT;
   max_range_squared = max_range_limit * max_range_limit;
   tow_ammo_type = munition_US_TOW;
}
void missile_tow_set_speed_factor( scale_speed )
REAL scale_speed;
   speed_factor = scale_speed;
}
void missile_tow_set_max_range_limit( limit_range )
REAL limit_range;
   max_range_limit = limit_range;
   max_range_squared = max_range_limit * max_range_limit;
}
void missile_tow_set_ammo_type( ammo )
ObjectType ammo;
{
    tow_ammo_type = ammo;
}
      *********
  ROUTINE: missile_tow_fire
                 tptr - A pointer to the TOW missile to be *
  PARAMETERS:
                  fired.
                 launch_point - The location in world
 * PARAMETERS:
                          coordinates that the missile is *
                          launched from.
            loc_sight_to_world - The sight to world
                                transformation matrix used *
                                only in this routine.
            launch_speed - The speed of the launch
                          platform (assumed to be in the
                          direction of the missile). *
            tube - The tube the missile was launched from.
 * RETURNS: none
   PURPOSE: This routine performs the functions
            specifically related to the firing of a TOW
            missile.
 ********
TOW_MISSILE *missile_tow_fire (tptr, launch_point, loc_sight_to_world,
        launch_speed, tube)
TOW_MISSILE *tptr;
VECTOR launch_point;
T_MATRIX loc_sight_to_world;
REAL launch_speed;
int tube;
```

```
{
                          /* Pointer to the particular generic missile
   MISSILE *mptr;
                             pointed at by _tptr_. */
 * Find _mptr_.
   mptr = &(tptr->mptr);
/*/
 * Set the initial time, location, orientation, and speed of the generic
 * missile.
/*/
   mptr->time = 0.0;
    vec_copy (launch_point, mptr->location);
   mat_copy (loc_sight_to_world, mptr->orientation);
    mptr->speed = launch_speed +
        (speed_factor * missile_util_eval_poly (TOW_BURN_SPEED_DEG,
                                               tow_burn_speed_coeff, 0.0));
    mptr->init_speed = launch_speed;
/*/
    Set the wire as uncut.
    tptr->wire_is_cut = FALSE;
    Tell the rest of the world about the firing of the missile. If this
   cannot be done, return.
/*/
    if (!missile_util_comm_fire_missile (mptr, MSL_TYPE_MISSILE,
            map_get_ammo_entry_from_network_type (tow_ammo_type),
            tow_ammo_type, tow_ammo_type, NULL, targetUnknown,
            objectIrrelevant, tube))
        return;
/*/
    If all was successful, set the missile state to TRUE and return.
    mptr->state = TRUE;
    return;
/***********************
  * ROUTINE: missile_tow_fly
                  tptr - A pointer to the TOW missile that is to *
  * PARAMETERS:
                   be flown out.
            sight_location - The location in world
                             coordinates of the gunner's
                             sight.
            loc_sight_to_world - The sight to world
                                 transformation matrix used *
                                 only in this routine.
  * RETURNS: none
  * PURPOSE: This routine performs the functions
            specifically related to the flying a TOW *
            missile.
```

```
void missile_tow_fly (tptr, sight_location, loc_sight_to_world)
TOW MISSILE *tptr;
VECTOR sight_location;
T_MATRIX loc_sight_to_world;
                         /* A pointer to the generic aspects of _tptr_. */
    MISSILE *mptr;
                          /* The current time after launch (ticks). */
    REAL time;
/*/
 * Set _mptr_ and _time_. These values are created mostly for increased
 * readablity.
/*/
    mptr = &(tptr->mptr);
    time = mptr->time;
/*/
  If the missile has reached its maximum range (not the maximum distance
 * its allowed to fly), cut the wire.
/*/
#ifdef notdeff
    if ((time > TOW_RANGE_LIMIT_TIME) && !tptr->wire_is_cut)
        tptr->wire_is_cut = TRUE;
#endif
    if (!tptr->wire_is_cut &&
        ((time > TOW_RANGE_LIMIT_TIME)
                                        11
         (max_range_limit > 0 &&
          kinematics_range_squared (veh_kinematics, mptr->location) >
          max range_squared) ))
        tptr->wire_is_cut = TRUE;
/*/
    Find the current missile speed and the cosines of the maximum allowed turn
    angles in each direction. The equations used are different before and
    after motor burnout.
/*/
    if (time < TOW_BURNOUT_TIME)</pre>
        mptr->speed = mptr->init_speed +
             (speed_factor *
             missile_util_eval_poly (TOW_BURN_SPEED_DEG,
                                      tow_burn_speed_coeff, time));
        missile_util_eval_cos_coeff (mptr, &tow_burn_turn_coeff, time);
     }
    else
         mptr->speed = mptr->init_speed +
             (speed_factor *
             missile_util_eval_poly (TOW_COAST_SPEED_DEG,
                                      tow_coast_speed_coeff, time));
         missile_util_eval_cos_coeff (mptr, &tow_coast_turn_coeff, time);
     }
 /*/
  * If the wire has been cut, set the ground as the target; otherwise,
  * find a target point which will fly the missile along the gunner's line of
    sight. This targeting scheme takes into account the errors introduced by
  * attempting to guide the missile in a canted position.
 /*/
```

```
if (tptr->wire_is_cut)
       missile_target_ground (mptr);
   else
       missile_target_level_los (mptr, sight_location, loc_sight_to_world);
/*/
  Try to actually fly the missile. If this fails stop the missile altogether
   and return.
/*/
   if (!missile_util_flyout (mptr))
       missile_tow_stop (tptr);
       return;
   }
   else
/*/
       If the missile successfully flew, check for an intersection with the
       ground or a vehicle. If one is found, blow up the missile, stop its
       flyout and return.
/*/
       if (missile_util_comm_check_intersection (mptr, MSL_TYPE_MISSILE))
           missile_util_comm_check_detonate (mptr, MSL_TYPE_MISSILE);
           missile_tow_stop (tptr);
           return;
        }
    }
   If the missile is to continue to fly, return.
/*/
    return;
/************************
 * ROUTINE: missile_tow_stop
              tptr - A pointer to the TOW missile that is to \star
 * PARAMETERS:
                  be stopped.
 * RETURNS: none
 * PURPOSE: This routine causes all concerned to forget
            about the missile. It should be called when
            the flyout of any TOW missile is stopped *
            (whether or not it has exploded). Note that
            this routine can only be called within this
            module.
 ***********************************
static void missile_tow_stop (tptr)
TOW_MISSILE *tptr;
{
 /*/
  * Tell the world to stop worrying about this missile then release the
 * memory for use by other missiles.
 /*/
```

```
missile_util_comm_stop_missile (&(tptr->mptr), MSL_TYPE_MISSILE);
   tptr->mptr.state = FALSE;
}
/***********************
 * ROUTINE: missile_tow_cut_wire
* PARAMETERS: tptr - A pointer to the TOW missile whose wire *
          is to be cut.
 * RETURNS: none
 * PURPOSE: This routine sets a flag indicating that the
         guidance wire of this missile is cut.
 ************************************
void missile_tow_cut_wire (tptr)
TOW_MISSILE *tptr;
/*/
* If the the wire is not already cut, cut the wire.
   if (!tptr->wire_is_cut)
       tptr->wire_is_cut = TRUE;
}
```

## APPENDIX M - rkt\_hydra.c

## Appendix M - Source code listing for rkt\_hydra.c.

The following appendix contains the source code listing for rkt\_hydra.c for convenience in document maintenance and understanding of the CSU.

```
/* $Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/libmissile/RCS/rkt_hydr
a.c,v 1.4 1993/01/28 23:27:59 cm-adst Exp $ */
 * $Log: rkt_hydra.c,v $
 * Revision 1.4 1993/01/28 23:27:59 cm-adst
 * P. DesMeules's changes for spcr 31
 * Revision 1.3 1993/01/06 21:19:06 cm-adst
 * R.Branson's changes for the weapons model.
 * Revision 1.1 1992/09/30 16:39:52 cm-adst
 * Initial Version
*/
static char RCS_ID[] = "$Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/li
bmissile/RCS/rkt_hydra.c,v 1.4 1993/01/28 23:27:59 cm-adst Exp $";
/******************************
* Revisions:
                                                           SP/CR Number
                                    Title
                           Author
                 Date
     Version
                 10/23/92 R. Branson Data File Initiali-
     1.2
                                        zation
                 10/30/92 R. Branson Added pathname to data
     1.3
                                        directory
                  11/25/92 R. Branson Changed %i to %d
     1.4
                  01/19/93 P.Desmeules Increased the size of the
                                                                  31
     1.5
                                        fgets to make sure the
                                        whole line is read in.
                      Description of Modification
       SP/CR No.
                      Hard coded defines changed to array elements.
                      Characteristics/parameter data array added.
                      Added file reads for rocket characteristics/
                         parameters.
                      Added "/simnet/data/" to each data file pathname.
      *********************************
 /***********************
  * FILE:
             rkt_hydra.c
             Kris Bartol
  * AUTHOR:
  * MAINTAINER: Kris Bartol
```

```
This file contains routines which govern
* PURPOSE:
                 the behavior of an Hydra70 Rocket flown with
                 a ballistic trajectory.
                10/06/90 kris
* HISTORY:
* Copyright (c) 1989 BBN Systems and Technologies, Inc.
* All rights reserved.
*****************************
#include "stdio.h"
#include "math.h"
#include "sim_types.h"
#include "sim_dfns.h"
#include "basic.h"
#include "mun_type.h"
#include "librva.h"
#include "libmap.h"
#include "libmatrix.h"
#include "libmiss_dfn.h"
#include "libmiss_loc.h"
#include "libmissile.h"
#include "rkt_hydra.h"
* Debug macro
*/
#ifdef FILEDBG
#define P(a)
#else
#define P(a)
#endif
                                       /* debugging is ON */
#define DEBUG
                                        "/simnet/data/hydra70.sd"
 #define HYDRA_TRAJ_FILE
                                        "/simnet/data/hydra70.sp"
 #define HYDRA_PARAM_FILE
 /*-- Define rocket performance characteristics --*/
 #define HYDRA_MAX_RANGE_M151 rkt_hydra_char[9]
#define HYDRA_MAX_RANGE_M261 rkt_hydra_char[10]
#define HYDRA_MAX_RANGE_M255 rkt_hydra_char[11]
 /*-- Define the states of an HYDRA70_ROCKET --*/
 #define HYDRA_FREE 0 /* Rocket available to launch */
 #define HYDRA_DETONATE 2 /* Rocket flying */
#define HYDRA_DETONATE 2 /* Rocket detonates - release or impact */
#define HYDRA_FALL 3 /* Sub-munitions falling.... */
#define HYDRA_RELEASED 4 /* Sub-munitions released towards impact */
#define HYDRA_REMOVE 10 /* Rocket gets killed at end of this tick */
                                      /* Sub-munitions released towards impact */
                                       /st Rocket gets killed at end of this tick st/
```

```
static REAL rkt_hydra_char[12] =
 M151_BURST_SPREAD, /* twin bursts are 3 m apart */
M261_BURST_HEIGHT, /* release submunitions 180 ft */
M261_BURST_RANGE, /* 0 m in front of target (49 ?) */
M261_BURST_SPREAD, /* twin bursts are 13 m apart */
M255_BURST_RANGE, /* release darts 150 m front of tgt */
M255_BURST_SPREAD, /* twin bursts are 35 m apart */
FLECH 60 MAY RANGE /* darts fly total of 750 m */
  FLECH_60_MAX_RANGE, /* darts fly total of 750 m
            /* hydra minimum range */

/* hydra maximum range for Soviet S-5 57mm Rocket */

/* hydra maximum range for _M151 [actual 9000 m] */

/* hydra maximum range for M261 */

/* hydra maximum range for M255 */
     50.0,
  5000.0,
  7000.0,
  7000.0,
  3200.0
};
/*-- burst releases 9 bombletts --*/
static int m73_per_m261_burst = M73_PER_M261_BURST;
/*-- pointer_to & number_of HYDRA70_ROCKET array --*/
static HYDRA_ROCKET *hydra_array; /* A pointer to Hydra70_Rkt memory */
                                                /* The number of defined missiles */
static int num_hydra;
/*-- array of pointers to Hydra70_Rockets in flight --*/
static HYDRA_ROCKET *hydra_fly[MAX_HYDRA70_ROCKET];
static int rkts_in_flight;
 /*-- Ballistics Table ... array of structures _MISSILE_BALLISTIC_OFFSETS_ --*/
 static MISSILE_BALLISTIC_OFFSETS ball_table[BALLISTIC_TABLE_SIZE];
 static int table_size;
 static BOOLEAN ball_table_loaded = FALSE;
 static VehicleID null_vehicleID;
 static int flight_time; /* Time Of Flight for ballistic traj */
 static REAL
                                                                                          */
      max_range_limit, /* [ MISSILE_US_MAX_RANGE_LIMIT ]
      speed_factor, /* [ MISSILE_US_SPEED_FACTOR ]
                                                                                          */
                                                                                          */
                             /* [0.0] <xyz> position offset of pylon
      pylon_x,
                              /* [0.0] */
      pylon_y,
                              /* [0.0] */
      pylon_z;
 static int flechette_veh_list; /* list ID of flechette target vehicles */
 static void missile_hydra_stop ();
 static void missile_hydra_purge_free_missiles ();
  /************************
   * ROUTINE: missile_hydra_init
   * PARAMETERS: rocket_array - Array of rockets of structure
                              type _HYDRA_ROCKET_
                   num_rockets - The number rockets defined in *
                                  _rockets_array_.
   * RETURNS: none
```

```
* PURPOSE: This routine copies the parameters into
           variables static to this module and initializes *
           the state of all the rockets.
 **********
void missile_hydra_init( rocket_array, num_rocket )
HYDRA_ROCKET *rocket_array;
int num_rocket;
      int
            i;
           data_tmp_int;
      int
      float data_tmp;
      char descript[80];
      FILE *fp;
      P(printf("$$$$ HYDRA rocket file data $$$$\n"););
/* DEFAULT CHARACTERISTICS DATA FOR rkt_hydra.c READ FROM FILE
                                                                        */
      fp = fopen("/simnet/data/rkt_hydr.d", "r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/rkt_hydr.d\n");
            exit();
      }
      rewind(fp);
            Read array data */
      fscanf(fp,"%d", &data_tmp_int);
      m73_per_m261_burst = data_tmp_int;
      fgets(descript, 80, fp);
      P(printf("m73_per_m261_burst is%3d %s", m73_per_m261_burst,
             descript););
      i=0;
      while(fscanf(fp, "%f", &data_tmp) != EOF){
            rkt_hydra_char[i] = data_tmp;
             fgets(descript, 80, fp);
            P(printf("rkt_hydra_char(%3d) is%11.3f %s", i,
                  rkt_hydra_char[i], descript););
             ++i;
      }
       fclose(fp);
 /* END DEFAULT CHARACTERISTICS DATA FOR rkt_hydra.c READ FROM FILE
     hydra_array = rocket_array;
     num_hydra = num_rocket < MAX_HYDRA70_ROCKET ?</pre>
         num_rocket : MAX_HYDRA70_ROCKET;
     for (i = 0; i < MAX_HYDRA70_ROCKET; i++)</pre>
         hydra_array[i].bmptr.state = HYDRA_FREE;
         hydra_array[i].bmpar.missile_id = 0;
```

```
rkts_in_flight = 0; /* no missiles in flight */
  for( i = 0; i < MAX_HYDRA70_ROCKET; i++ )
      hydra_fly[i] = 0;
  pylon_x = 0.0;
  pylon_y = 0.0;
   pylon_z = 0.0;
   flight_time = 0;
   speed_factor = MISSILE_US_SPEED_FACTOR;
  max_range_limit = MISSILE_US_MAX_RANGE_LIMIT;
   if (!ball_table_loaded)
    {
* load Hydra70 Rocket's ballistic table
   printf( "loading Hydra70 Rocket's ballistic table %s\n",
         HYDRA_TRAJ_FILE );
   table_size =
       missile_util_load_ball_traj_file( HYDRA_TRAJ_FILE, ball_table );
     ball_table_loaded = TRUE;
     }
 * create _flechette_veh_list_ for proximity fuze
   flechette_veh_list = rva_create_output_list( flechette_is_valid_veh );
#ifdef notdef
   flechette_veh_list = RVA_ALL_VEHICLES_LIST;
#endif
* initialize the proximity fuze for rockets armed with Flechette's
   missile_fuze_prox_init();
int missile_hydra_is_free( rocket )
int rocket;
   return( (hydra_array[rocket].bmptr.state == HYDRA_FREE ));
}
            **************
 * ROUTINE: missile_hydra_set_pylon_position_offsets
 * PARAMETERS: x = X offset (in meters ) from center of HULL.
               y = Y 	ext{ offset.}
               z = Z offset.
              none.
 * RETURNS:
              Sets the X, Y and Z offsets from center of
 * PURPOSE:
              HULL for trajectory calculations.
void missile_hydra_set_pylon_position_offsets( x, y, z )
REAL x, y, z;
```

```
{
   pylon_x = x;
   pylon_y = y;
   pylon_z = z;
}
void missile_hydra_set_speed_factor( speed_scale )
REAL speed_scale;
    speed_factor = speed_scale;
}
void missile_hydra_set_max_range_limit( limit_range )
REAL limit_range;
    max_range_limit = limit_range;
/************************
 * ROUTINE: missile_hydra_set_pylon_articulation
                tgt_range - Range to target.
 * PARAMETERS:
               rkt_type - Type of Rocket to be launched.
               time - Pointer to Time Of Flight
                       variable in vehicle-spec code. [int]
               se_angle - Pointer to Super Elevation
                       variable in vehicle-spec code. [REAL]
               lead_angle - Pointer to Lead Elevation
                       variable in vehicle-spec code. [REAL]
 * RETURNS:
               none.
               Sets _laser_range_ of next Hydra70 rocket to
  PURPOSE:
               be launched and calculates Time Of Flight,
               Super Elevation angle and Lead angle for next
               rocket launch.
           *********************
void missile_hydra_set_pylon_articulation( tgt_range, rkt_type, time,
                                         se_angle, lead_angle )
        tgt_range;
REAL
        rkt_type, *time;
int
        *se_angle, *lead_angle;
REAL
                      /* Range to target */
    REAL range;
    REAL ball_range; /* Range to look-up in Ballistic Table */
    if( tgt_range < HYDRA_MIN_RANGE )</pre>
        range = HYDRA_MIN_RANGE;
    else if(( max_range_limit > 0.0 ) &&
            ( tgt_range > max_range_limit ) )
        range = max_range_limit;
    else
        range = tgt_range;
 /* SuperElevation & TOF for each Rocket Type */
     switch( rkt_type )
    `{
```

```
/* type 101b WARHEAD */
  case ROCKET_HE:
      if( range > HYDRA_MAX_RANGE_M151 )
          range = HYDRA_MAX_RANGE_M151;
      ball_range = range / speed_factor;
      missile_util_ballistics_calc_traj( ball_table, table_size,
                                        ball_range, 0.0, 0.0,
                                        time, se_angle );
      *lead_angle = atan( (rkt_hydra_char[ 0] - pylon_x) / range );
                      /* Does not have a timed fuze */
      *time = -5;
      break;
                                              /* type MPSM */
   case ROCKET_MPSM:
       if( range > HYDRA_MAX_RANGE_M261 )
          range = HYDRA_MAX_RANGE_M261;
      ball_range = range / speed_factor;
      missile_util_ballistics_calc_traj( ball_table, table_size,
                                        ball_range, 0.0, rkt_hydra_char[ 1],
                                         time, se_angle );
       *lead_angle = atan( (rkt_hydra_char[ 3] - pylon_x) / range );
       break:
                                               /* type FLECHETTE */
   case ROCKET_FLECHETTE:
       if( range > HYDRA_MAX_RANGE_M255 )
           range = HYDRA_MAX_RANGE_M255;
       ball_range = range / speed_factor;
       missile_util_ballistics_calc_traj( ball_table, table_size,
                                         ball_range, rkt_hydra_char[ 4], 0.0,
                                         time, se_angle );
       *lead_angle = atan((rkt_hydra_char[ 5] - pylon_x) /
                          (range - rkt_hydra_char[ 4]));
       break:
       printf( "hydra_set_pylon_articul: unknown warhead_type %d\n", rkt_type );
   default:
       *time = 0;
       *se\_angle = 0.0;
       *lead_angle = 0.0;
       break;
   flight_time = *time;
}
/*****************
 * ROUTINE: missile_hydra_fire
                 rkt_type - Type of Rocket warhead.
 * PARAMETERS:
                ammo - Ammo Type of rocket's warhead.
                launch_pt - The location in world
                           coordinates that the rocket is
                           launched from.
            launch_orient - The sight to world
                           transformation matrix used *
                           only in this routine.
                launch_speed - Speed of launch platform
                                (assumed to be in the direction *
                                of the Rocket).
 * RETURNS: TRUE if successful, FALSE if not.
```

```
* PURPOSE: This routine performs the functions
           specifically related to the firing of a HYDRA70 *
    rocket.
int missile_hydra_fire( rkt_type, ammo, launch_pt,
                      launch_orient, launch_speed )
int rkt_type;
ObjectType ammo;
VECTOR launch_pt;
T_MAT_PTR launch_orient;
REAL launch_speed;
    T_MATRIX
        launch_lead,
        launch_se;
    REAL
                       /* munition_specific SuperElevation angle */
       se_angle,
                       /* munition_specific (+/-)Lead angle */
        lead_angle;
                       /* munition_specific FlightTime */
    int time;
    HYDRA_ROCKET *rkt;
    BALLISTIC_MISSILE *bmptr;
    ObjectType fuze;
    int i, valid_msl;
/* get next FREE rocket */
    valid_msl = 0;
    rkt = hydra_array;
    for( i = 0; i < MAX_HYDRA70_ROCKET; i++, rkt++ )
        if( rkt->bmptr.state == HYDRA_FREE )
            valid_msl = 1;
            hydra_fly[rkts_in_flight] = rkt;
            bmptr = &(rkt->bmptr);
 #if DEBUG
            printf( "Launching Rocket %d\n", i );
                                        /* rkts_in_flight == # flying */
 #endif
            rkts_in_flight++;
            break;
                               /* no available missile to launch */
     if(!valid_msl)
         return( FALSE );
 /* set MaxRange for Rocket Type */
     switch( rkt_type )
                                        /* High Explosive */
     case ROCKET_HE:
         bmptr->max_range = HYDRA_MAX_RANGE_M151;
         rkt->sub_mun_type = SUB_MUN_NONE;
         rkt->sub_ammo_type = 0;
         fuze = munition_US_M433;
         break;
                                       /* Multi-Purpose Sub-Munition */
     case ROCKET_MPSM:
```

```
bmptr->max_range = HYDRA_MAX_RANGE_M261;
      rkt->sub_mun_type = SUB_MUN_IMPACT;
       rkt->sub_ammo_type = munition_US_M73;
      rkt->sub_munition.impact.ammo = munition_US_M73;
       rkt->sub_munition.impact.fuze = munition_US_M433;
       rkt->sub_munition.impact.quantity = m73_per_m261_burst;
       rkt->sub_munition.impact.height = rkt_hydra_char[ 1];
       fuze = munition_US_M439;
       break;
                                       /* Flechette discharging warhead */
   case ROCKET_FLECHETTE:
       bmptr->max_range = HYDRA_MAX_RANGE_M255;
       rkt->sub_mun_type = SUB_MUN_CANISTER;
       rkt->sub_ammo_type = munition_US_Flechette_60;
       rkt->sub_munition.dart.ammo = munition_US_Flechette_60;
       rkt->sub_munition.dart.fuze = 0;
       fuze = munition_US_M439;
       break:
   default:
       printf( "hydra_fire_rkt: unknown rocket_type %d\n", rkt_type );
       rkts_in_flight--;
       bmptr -> state = HYDRA_FLY;
       return ( FALSE );
       break;
   }
   mat_copy( launch_orient, bmptr->launcher_C_world );
   mat_copy( launch_orient, bmptr->orientation );
   vec_copy( launch_pt, bmptr->location );
   bmptr->speed = launch_speed;
/\star -- Tell the rest of the world about the firing of this B-missile. --
* -- If this cannot be done, return FALSE. --
 */
   if( !missile_util_comm_fire_missile
       ( bmptr, MSL_TYPE_BALLISTIC,
       map_get_ammo_entry_from_network_type( ammo ),
       ammo, ammo, /*guises*/
       &(null_vehicleID), 0/*targ_type*/, fuze, 0/*tube*/ ))
       rkts_in_flight--;
       bmptr -> state = HYDRA_FLY;
       return ( FALSE );
    }
    bmptr -> max_flight_time = flight_time;
    bmptr -> ammo_type = ammo;
                                       /* initialize in-flight timer */
    bmptr -> time = 0;
                                      /* first point into Ball-table */
    bmptr -> ball_index = 0;
                                       /* rocket is now flying */
    bmptr -> state = HYDRA_FLY;
    return( TRUE );
}
/******************************
                missile_hydra_fly_rockets
 * ROUTINE:
 * PARAMETERS: none
 * RETURNS:
              none
```

```
* PURPOSE: This routine flys out all rockets that are in
              a flying state.
               ***********************************
void missile_hydra_fly_rockets()
    register int i;
    int at_least_one_empty_MPSM;
        Fly out all launched & flying rockets.
        -- may have to also 'fly out' all released submunitions --
    at_least_one_empty_MPSM = FALSE;
    for( i = 0; i < rkts_in_flight; i++ )</pre>
        switch( hydra_fly[i]->bmptr.state )
        case HYDRA_FREE:
            hydra_fly[i]->bmptr.state = HYDRA_REMOVE;
            break;
        case HYDRA_FLY:
            missile_hydra_fly( hydra_fly[i] );
            break:
        case HYDRA_DETONATE:
            switch( hydra_fly[i]->sub_ammo_type )
                                                /* MPSM bomblets */
            case munition_US_M73:
                missile_m73_init
                     ( &(hydra_fly[i]->bmptr),
                     &(hydra_fly[i]->sub_munition),
                       ball_table[ hydra_fly[i]->bmptr.ball_index ].speed );
                hydra_fly[i]->bmptr.state = HYDRA_FALL;
                break;
                                               /* FLECHETTE darts */
             case munition_US_Flechette_60:
                missile_flechette_init
                     ( &(hydra_fly[i]->bmptr),
                      &(hydra_fly[i]->sub_munition),
                      ball_table[ hydra_fly[i]->bmptr.ball_index ].speed );
                hydra_fly[i]->bmptr.state = HYDRA_RELEASED;
                 break:
             default:
                 printf( "Hydra_Detonate: R_%d unknown ammo-type\n",i );
                 missile_hydra_stop( hydra_fly[i] );
                 break;
             break;
         case HYDRA_FALL:
             switch( hydra_fly[i]->sub_ammo_type )
                                                  /* type MPSM */
             case munition_US_M73:
                 if( missile_m73_drop( &(hydra_fly[i]->bmptr),
                                       &(hydra_fly[i]->sub_munition)))
                     hydra_fly[i]->bmptr.state = HYDRA_RELEASED;
                 break;
             default:
```

```
printf( "Hydra_Fall(): R_%d bad sub_munition\n",i );
              missile_hydra_stop( hydra_fly[i] );
              break;
         break;
      case HYDRA_RELEASED:
          switch( hydra_fly[i]->sub_ammo_type )
                                              /* type MPSM */
          case munition_US_M73:
              if( ! missile_m73_impact( &(hydra_fly[i]->bmptr),
                                       &(hydra_fly[i]->sub_munition)))
              {
                  at_least_one_empty_MPSM = TRUE;
                  missile_hydra_stop( hydra_fly[i] );
              }
              break;
                                              /* type FLECHETTE */
          case munition_US_Flechette_60:
              if( ! missile_flechette_fly( &(hydra_fly[i]->bmptr),
                                      &(hydra_fly[i]->sub_munition),
                                      flechette_veh_list ))
               {
                  missile_hydra_stop( hydra_fly[i] );
                  missile_fuze_prox_stop
                       ( &(hydra_fly[i]->sub_munition.dart.pptr) );
               break;
           default:
               printf( "Hydra_Release: R_%d bad sub_munition\n",i );
               missile_hydra_stop( hydra_fly[i] );
               break;
           }
           break;
       case HYDRA_REMOVE:
           break;
       default:
           printf( "Msl_hydra_fly_rkts(): rkt_%d not flying\n", i );
           missile_hydra_stop( hydra_fly[i] );
           break;
       }
/* Send out remaining (if any) Indirect Fire pkts */
    if( at_least_one_empty_MPSM )
       network_ifire_send_indirect_fire();
/* Get rid of DEAD rockets */
    missile_hydra_purge_free_missiles();
}
                                  *****
/********
               missile_hydra_fly
 * ROUTINE:
                 rkt - Pointer to a _HYDRA_ROCKET_ structure
 * PARAMETERS:
               none
 * RETURNS:
 * PURPOSE: This routine performs the functions
            specifically related to the flying an HYDRA70
```

```
and frees up the Rocket for another launch.
static void missile_hydra_stop( rkt )
HYDRA_ROCKET *rkt;
                       *bmptr;
   BALLISTIC_MISSILE
   int i;
   bmptr = &( rkt->bmptr );
 * Tell the world to stop worrying about this missile then release the
 * memory for use by other missiles.
   missile_util_comm_stop_missile( bmptr, MSL_TYPE_BALLISTIC );
#if DEBUG
    bmptr->time, bmptr->missile_id, bmptr->location[0],
          bmptr->location[1], bmptr->location[2] );
#endif
 * Mark rocket to be Removed
    bmptr->state = HYDRA_REMOVE;
}
static void missile_hydra_purge_free_missiles()
    int i;
    i = 0;
    while( i < rkts_in_flight )</pre>
        if( hydra_fly[i]->bmptr.state == HYDRA_REMOVE )
             * Swap --BAD-- rocket[i] with --LAST-- rocket[rkts_in_flight]
             * Cut-off (now BAD) --LAST-- rocket
             * Check (now Good) rocket[i]
             */
            hydra_fly[i]->bmptr.state = HYDRA_FREE;
            rkts_in_flight--;
            hydra_fly[i] = hydra_fly[rkts_in_flight];
            hydra_fly[rkts_in_flight] = 0;
        }
        else
             * Check next rocket[i+1]
             */
            i++;
 }
```

```
void mbmat( mat )
T_MAT_PTR mat;
    int i, j;
    for( i=0; i<3; i++ )
        for( j=0; j<3; j++ )
            printf( " %1.4lf ", mat[i][j] );
        printf( "\n" );
    }
}
void mbmat_nan( mat )
T_MAT_PTR mat;
    int i, j;
    union foo
        REAL df;
        long 1[2];
    } x;
    for( i=0; i<3; i++)
         for( j=0; j<3; j++ )
    printf( " %1.41f ", mat[i][j] );</pre>
         printf( "-->" );
         for( j=0; j<3; j++)
             x.df = mat[i][j];
             printf( " 0x%08x 0x%08x", x.1[0], x.1[1] );
         printf( "\n" );
    }
}
void mbm( n, msg )
int n;
char msg[];
    printf( "BM: %d -> %s\n", n, msg );
void mbfl( n, msg )
REAL n;
char msg[];
    printf( "BM: %6.41f -> %s\n", n, msg );
}
```

# Appendix N - Source code listing for rwa\_hydra.c.

The following appendix contains the source code listing for rwa\_hydra.c for convenience in document maintenance and understanding of the CSU.

```
/* SHeader: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/rwa/src/RCS/rwa_hydra.c,v 1.4
1993/01/28 23:33:00 cm-adst Exp $ */
 * $Log: rwa_hydra.c,v $
 * Revision 1.4 1993/01/28 23:33:00 cm-adst
 * P. DesMeueles's changes for spcr 31
 * Revision 1.3 1993/01/06 21:29:20 cm-adst
 * R.Branson's changes for the weapons model.
 * Revision 1.1 1992/09/30 17:02:58 cm-adst
 * Initial Version
*/
static char RCS_ID[] = "$Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/rwa/src/R
CS/rwa_hydra.c,v 1.4 1993/01/28 23:33:00 cm-adst Exp $";
* Revisions:
                                                            SP/CR Number
                                    Title
                           Author
     Version
                 Date
                  10/23/92 R. Branson Data File Initiali-
     1.2
                                         zation
                 10/30/92 R. Branson Added pathname to data
     1.3
                                         directory
                  01/19/93 P.Desmeules Increased the size of the
      1.5
                                        fgets to make sure the
                                        whole line is read in.
                      Description of Modification
        SP/CR No.
                      Hard coded defines changed to array elements.
                       Characteristics/parameter data array added.
                       Added file reads for hydra rocket characteristics/
                         parameters.
                       Added "/simnet/data/" to each data file pathname.
                     ******************
 /***********************
  * SYSTEM NAME: rwa
             rwa_hydra.c
  * FILE:
              Kris Bartol
  * AUTHOR:
  * SIMNET simulation of Hydra70 Rocket
```

```
* Copyright (c) 1990 BBN Advanced Simulation Division.
* All rights reserved.
******************
#include "simstdio.h"
#include "sim_types.h"
#include "sim_dfns.h"
#include "sim_macros.h"
#include "basic.h"
#include "mun_type.h"
#include "veh_type.h"
#include "libmatrix.h"
#include "libmath.h"
#include "librotate.h"
#include "libturret.h"
#include "libhull.h"
#include "libkin.h"
#include "libcig.h"
#include "libimps.h"
#include "libmap.h"
#include "libmissile.h"
#include "libmiss_dfn.h"
#include "rkt_hydra.h"
#include "rwa_kinemat.h"
#include "rwa_weapons.h"
#include "rwa_meter.h"
#include "rwa_config.h"
* Debug macro
#ifdef FILEDBG
#define P(a)
#else
#define P(a)
#endif
                      /* debugging is ON */
#define DEBUG
                0
                0
#define LEFT
#define RIGHT
#define NUM_ROCKETS_LAUNCHED_PER_TICK
 * Define rocket characteristics.
/*/
#define HYDRA_LAUNCHER_POS_X hydra_rkt_char[0]
#define HYDRA_LAUNCHER_POS_Y hydra_rkt_char[1]
 #define HYDRA_LAUNCHER_POS_Z hydra_rkt_char[2]
```

```
/* *****
* Articulation Limits are +4 to -15 degrees but are adjusted to
* +19 to -15 degrees for simulation's fixed OTW reticle
* *****/
                              ( mil_to_rad(hydra_rkt_char[3]))
#define SOVIET_ARTICULATION
                               ( deg_to_rad(hydra_rkt_char[4]))
#define HULL_NEG_5_PITCH
                               ( deg_to_rad(hydra_rkt_char[5]))
#define ARTICULATION_MAX
                                ( deg_to_rad(hydra_rkt_char[6]))
#define ARTICULATION_MIN
/*/
 * Hydra rocket characteristic parameters initialized to default values.
static REAL hydra_rkt_char[7] =
           /* hydra launcher position X */
    4.5,
           /* hydra launcher position Y */
    0.5,
           /* hydra launcher position Z */
   -2.0,
           /* mils of Soviet articulation */
  104.0,
          /* degrees of hull negative pitch */
   -5.0,
          /* degrees of maximum articulation */
   19.0,
          /* degrees of minimum articulation */
  -15.0
};
ROTATE_ELEMENT_DEF (articulation_element);
ROTATE_ELEMENT_DEF (pylon_L_element);
ROTATE_ELEMENT_DEF (pylon_R_element);
static HYDRA_ROCKET hydras[MAX_HYDRA70_ROCKET + 1] = { 0 };
static VehicleID null_VehicleID;
                                /* Time Of Flight for ballistic traj */
static int flight_time;
static REAL
                                /* Adj angle for ballistic traj */
    super_elevation,
                                /* Range by which to calculate ballistics */
    target_range;
static ObjectType ammo_type; /* Ammo_Type of rockets to be launched */
                                /* one of [ HE | MPSM | FLECHETTE ]
static int warhead_class;
                                /* TRUE when pylon articulation is complete */
static int pylons_set;
static int left_rocket_launch; /* TRUE --> launch left rocket */
static int right_rocket_launch; /* TRUE --> launch right rocket */
static VECTOR left_launcher_pos = { 4.5, 0.0, 0.0 };
 static VECTOR right_launcher_pos = { 4.5, 0.0, 0.0 };
 static VECTOR articulation_pos = { 0.0, 0.5, -2.0 };
 extern REAL weapons_get_rocket_range();
 extern REAL kinematics_get_true_airspeed();
 extern void mbmat();
 extern void mbmat_nan();
 extern void mbvec();
 ROTATE_ELEMENT *articulation()
 {
```

```
return( &articulation_element );
}
ROTATE_ELEMENT *pylon_L()
    return( &pylon_L_element );
}
ROTATE_ELEMENT *pylon_R()
    return( &pylon_R_element );
}
void hydra_launch_rocket_left()
    left_rocket_launch = TRUE;
void hydra_launch_rocket_right()
    right_rocket_launch = TRUE;
}
int hydra_launch_rocket( launch_from_right )
int launch_from_right; /* 0 = left-side (neg) :: 1 = right-side (pos) */
    T_MAT_PTR launch_orient;
    VECTOR launch_velocity;
    REAL
         *launch_point,
         se_angle,
         lead_angle;
 /* get launch_point & launch_orient */
                             /* launch from right */
     if( launch_from_right )
     {
         launch_point = rotate_get_loc( world(), pylon_R() );
         launch_orient = rotate_get_mat( pylon_R(), world() );
     }
     else
         launch_point = rotate_get_loc( world(), pylon_L() );
         launch_orient = rotate_get_mat( pylon_L(), world() );
 #if DEBUG
     if( mat_check(launch_orient) == FALSE )
         mbmat_nan( launch_orient );
     if( !missile_hydra_fire( warhead_class, ammo_type,
                              launch_point, launch_orient,
                              (kinematics_get_true_airspeed()/15) /*init speed*/))
 #if DEBUG
         printf( "No memory in missile_comm for HYDRA\n");
```

```
#endif
        printf( "Rocket launch failed\n" ); .
       return( FALSE );
    return( TRUE );
}
int hydra_pylons_are_set()
    return( pylons_set );
}
void hydra_set_pylon_articulation( WAS_position )
int WAS_position;
{
    MUNITION_DATA *mun_data;
                       /* time of flight to fly _range_ meters */
    int flight_time;
    REAL
                        /* range to target */
        range,
                        /* super elevation angle for trajectory */
        super_elev,
                        /* dispersion angle for trajectory */
        dispersion;
 * Given _range_ & _ammo_type_ ::
        * calculate and return super_elev & dispersion angles
        * calculate and set Time-Of-Flight timer
        * set _ammo_type_ of next rocket(s) to be fired
    mun_data = rwa_config_get_was_munition_info (WAS_position);
    ammo_type = mun_data->munition_type;
    if (mun_data->code != MUNITION_ROCKET)
      /* bombs, for example */
      return;
     switch(mun_data->data.rocket.warhead)
     case WARHEAD_HE:
        warhead_class = ROCKET_HE;
         break;
     case WARHEAD_MPSM:
         warhead_class = ROCKET_MPSM;
         break;
     case WARHEAD_FLECHETTE:
         warhead_class = ROCKET_FLECHETTE;
         break:
     default:
         printf( "hydra_set_artic: unknown warhead %d for WAS %d\n",
                mun_data->data.rocket.warhead, WAS_position );
         break;
     }
 /*
  * Get rocket range & calculate SuperElevation and Dispersion angles
  */
```

```
pylons_set = FALSE;
   if( mun_data->data.rocket.articulation )
       range = weapons_get_rocket_range();
   else
       range = (REAL)(mun_data->data.rocket.flyout_range);
* Set pylon Super Elevation angle & pylon Dispersion angle
   missile_hydra_set_pylon_articulation( range, warhead_class, &flight_time,
                                         &super_elev, &dispersion );
   super_elev += HULL_NEG_5_PITCH;
   rotate_set_angle( articulation(), super_elev );
   rotate_set_angle( pylon_R(), (- dispersion) );
    rotate_set_angle( pylon_L(), dispersion );
}
void hydra_config_rockets()
    MUNITION_DATA *mun_data;
    int i;
    for( i = 0; i < MAX_WAS_POSITIONS; i++ )</pre>
        if( (mun_data = rwa_config_get_was_munition_info( i )) == NULL )
            continue;
        if( mun_data->code == MUNITION_ROCKET )
            missile_hydra_set_speed_factor
                 ( (REAL) (mun_data->data.rocket.speed_factor) );
            missile_hydra_set_max_range_limit
                 ( (REAL) (mun_data->data.rocket.flyout_range) );
        }
    }
}
void hydra_init ()
            i;
      int
            data_tmp_int;
      int
      float data_tmp;
      char descript[80];
      FILE *fp;
      P(printf("$$$$ HYDRA file data $$$$\n"););
/* DEFAULT CHARACTERISTICS DATA FOR rwa_hydra.c READ FROM FILE
                                                                           */
       fp = fopen("/simnet/data/rwa_hydr.d","r");
       if(fp==NULL){
             fprintf(stderr, "Cannot open /simnet/data/rwa_hydr.d\n");
             exit();
       }
       rewind(fp);
```

```
Read array data
  . /*
     i=0;
     while(fscanf(fp, "%f", &data_tmp) != EOF){
           hydra_rkt_char[i] = data_tmp;
           fgets(descript, 80, fp);
           P(printf("hydra_rkt_char(%3d) is%11.3f %s", i,
                 hydra_rkt_char[i], descript););
           ++i;
      }
      fclose(fp);
/* END DEFAULT CHARACTERISTICS DATA FOR rwa_hydra.c READ FROM FILE */
      left_launcher_pos[0] = HYDRA_LAUNCHER_POS_X;
      right_launcher_pos[0] = HYDRA_LAUNCHER_POS_X;
      articulation_pos[1] = HYDRA_LAUNCHER_POS_Y;
      articulation_pos[2] = HYDRA_LAUNCHER_POS_Z;
    if(!rotate_init_element( &articulation_element, hull(),
                            1.0, 0.0, 0.0, 0.0,
                        ARTICULATION_MIN, ARTICULATION_MAX, /*TWO_*/PI, /*rate*/
                        0.0, HYDRA_LAUNCHER_POS_Y, HYDRA_LAUNCHER_POS_Z ))
        printf( "Rotate_Init_Element: articulation_element FAILED\n" );
    rotate_init_element( &pylon_L_element, articulation(), 0.0, 0.0, 1.0, 0.0,
                        -TWO_PI, TWO_PI, TWO_PI, /*rate*/
                        -HYDRA_LAUNCHER_POS_X, 0.0, 0.0 );
    rotate_init_element( &pylon_R_element, articulation(), 0.0, 0.0, 1.0, 0.0,
                        -TWO_PI, TWO_PI, TWO_PI, /*rate*/
                        HYDRA_LAUNCHER_POS_X, 0.0, 0.0 );
    missile_hydra_init( hydras, MAX_HYDRA70_ROCKET );
    missile_hydra_set_pylon_position_offsets( HYDRA_LAUNCHER_POS_X,
                                               HYDRA_LAUNCHER_POS_Y,
                                               HYDRA_LAUNCHER_POS_Z );
    hydra_config_rockets();
    left_rocket_launch = FALSE;
    right_rocket_launch = FALSE;
    pylons_set = FALSE;
}
void hydra_simul()
    missile_hydra_fly_rockets();
    if( !pylons_set )
        pylons_set = TRUE;
         rotate_set_no_rotate( pylon_R() );
         rotate_set_no_rotate( pylon_L() );
         rotate_set_no_rotate( articulation() );
     }
```

```
else
       if( left_rocket_launch )
            if( hydra_launch_rocket( LEFT ) )
                left_rocket_launch = FALSE;
        if( right_rocket_launch )
            if( hydra_launch_rocket( RIGHT ) )
                right_rocket_launch = FALSE;
    }
}
void mbvec( str, vec )
char *str;
VECTOR vec;
{
    printf( "%s [ %1.4lf %1.4lf %1.4lf ]\n",
           str, vec[X], vec[Y], vec[Z] );
}
```

# Appendix O - Source code listing for sub\_flech.c.

The following appendix contains the source code listing for sub\_flech.c for convenience in document maintenance and understanding of the CSU.

```
/* $Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/libmissile/RCS/sub_flec
h.c,v 1.4 1993/01/28 23:27:09 cm-adst Exp $ */
 * $Log: sub_flech.c,v $
 * Revision \frac{1}{1}.4 1993/01/28 23:27:09 cm-adst
 * P.DesMeules's changes for spcr 31
 * Revision 1.3 1993/01/06 21:19:27 cm-adst
 * R.Branson's changes for the weapons model.
 * Revision 1.1 1992/09/30 16:39:52 cm-adst
 * Initial Version
*/
static char RCS_ID[] = "$Header: /a3/adst-cm/RWA/AIRNET/simnet/vehicle/libsrc/li
bmissile/RCS/sub_flech.c,v 1.4 1993/01/28 23:27:09 cm-adst Exp $";
   ****************
  Revisions:
                                                           SP/CR Number
                                    Title
                  Date
                         Author
     Version
                  10/23/92 R. Branson Data File Initiali-
                                       zation
                 10/30/92 R. Branson Added pathname to data
     1.3
                                       directory
                 11/25/92 R. Branson Changed %i to %d
      1.4
                  01/19/93 P.Desmeules Increased the size of the
      1.5
                                       fgets to make sure the
                                       whole line is read in.
                      Description of Modification
        SP/CR No.
                      Hard coded defines changed to array elements.
                      Characteristics/parameter data array added.
                      Added file reads for sub_flechette characteristics/
                         parameters and flechette speed coefficients.
                      Added "/simnet/data/" to each data file pathname.
       **************************
       ****************
            sub_flech.c
Kris Bartol
  * FILE:
  * AUTHOR:
  * MAINTAINER: Kris Barto:
```

```
This file contains routines which simulates
* PURPOSE:
              the behavior of sub-munitions of type
              munition_US_Flechette_60.
              10/06/90 kris
* HISTORY:
* Copyright (c) 1989 BBN Systems and Technologies, Inc.
* All rights reserved.
******************************
#include "stdio.h"
#include "math.h"
#include "sim_types.h"
#include "sim_dfns.h"
#include "basic.h"
#include "mun_type.h"
#include "libhull.h"
#include "libimps.h"
#include "libkin.h"
#include "libmath.h"
#include "libmap.h"
#include "libmatrix.h"
#include "libmiss_dfn.h"
#include "libmiss_loc.h"
#include "rkt_hydra.h"
* Debug macro
#ifdef FILEDBG
#define P(a)
#else
#define P(a)
#endif
                              /* debugging is ON */
#define DEBUG 0
#define INVEST_DIST_SQ
                            sub_flech_char[0]
                             sub_flech_char[1]
#define FUZE_DIST_SQ
                              sub_flech_poly_deg
#define FLECHETTE_SPEED_DEG
 * Sub_flechette characteristic parameters initialized to default values.
static REAL sub_flech_char[3] =
{
          /* (100 m)^2 :: max speed < 100 */
 10000.0,
   306.25, /* (17.5 m)^2 :: flechettes fly
                            in a cylinder with a radius
                           of 17.5 m and length of 750 m ^{*}/
                      /* darts fly total of 750m */
 FLECH_60_MAX_RANGE
};
```

```
* The following term sets the order of the polynomial used to determine
 * the speed of the flechettes.
/*/
static int sub_flech_poly_deg = 3;
   Coefficients for the speed polynomial for flechettes initialized
 * to default values.
/*/
static REAL flechette_speed_coef[5] =
                       /* a_0 - m/tick
    41.75,
     11.75,

-0.20397254, /* a_1 - m/tick/m

0.00022724278, /* a_2 - m/tick/m^2

-0.00000008633, /* a_3 - m/tick/m^3
    -0.20397254,
                                                */
    -0.00000008633,
     0.0
};
static VECTOR zero_vector = { 0.0, 0.0, 0.0 };
static VehicleID null_VehicleID;
/* this routine is invoked by the rva for each vehicle to see if it
 * should be included on the flechette valid vehicle list
 */
 flechette_is_valid_veh (veh)
 VehicleAppearanceVariant *veh;
    return( /* is_alive_vehicle (veh->appearance) */ TRUE );
 }
 /*****************
  * ROUTINE: missile_flechette_init
  * PARAMETERS: bmptr - Pointer to a _BALLISTIC_MISSILE_
                         structure that's ammo-type is Flechette *
                         i.e. it releases sub-munitions of type *
                         _munition_US_Flechette_60_.
                 sub_mun - Pointer to sub-munition structure
                         associated with _bmptr_.
              init_speed - Terminal speed of rocket ==
                         initial speed of flechettes.
  * RETURNS:
                 Initialize rocket's _mmptr_ to behave according *
  * PURPOSE:
                 sub-munitions type of
               _munition_US_Flechette_60_.
           *******
 void missile_flechette_init( bmptr, smb_mun, init_speed )
 BALLISTIC_MISSILE *bmptr;
                        *sub_mun;
 BALLISTIC_SUB_MUN
```

```
init_speed;
REAL
    BALLISTIC_CANISTER *dart;
    VECTOR velocity;
      int
            i;
      int
            data_tmp_int;
      float data_tmp;
      char descript[80];
      FILE *fp;
      P(printf("$$$$ FLECHETTE file data $$$$\n"););
/* DEFAULT CHARACTERISTICS DATA FOR sub_flech.c READ FROM FILE
                                                                          */
      fp = fopen("/simnet/data/sub_flec.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/sub_flec.d\n");
      }
      rewind(fp);
      /*
            Read array data
      i=0;
      while(fscanf(fp, "%f", &data_tmp) != EOF){
            sub_flech_char[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("sub_flech_char(%3d) is%11.3f %s", i, sub_flech_char[i],
                        descript););
            ++i;
      }
      fclose(fp);
                                                                          */
/* END DEFAULT CHARACTERISTICS DATA FOR sub_flech.c READ FROM FILE
                                                                          */
/* DEFAULT FLECHETTE SPEED DATA FOR sub_flech.c READ FROM FILE
      fp = fopen("/simnet/data/flec_spd.d", "r");
      if(fp==NULL){
             fprintf(stderr, "Cannot open /simnet/data/flec_spd.d\n");
            exit();
      }
      rewind(fp);
            Read degree of polynomial */
      fscanf(fp, "%d", &data_tmp_int);
      FLECHETTE_SPEED_DEG = data_tmp_int;
      fgets(descript, 80, fp);
      P(printf("sub_flech_poly_deg is%3d %s", FLECHETTE_SPEED_DEG,
                 descript););
             Read array data
                               */
      i=0;
```

```
while(fscanf(fp, "%f", &data_tmp) != EOF){
           flechette_speed_coef[i] = data_tmp;
           fgets(descript, 80, fp);
           P(printf("flechette_speed_coef(%3d) is%11.3f %s", i,
                      flechette_speed_coef[i], descript););
           ++i;
     }
     fclose(fp);
/* END DEFAULT BURN SPEED DATA FOR sub_flech.c READ FROM FILE
   bmptr->time = 0;
   dart = &(sub_mun->dart);
   dart->distance = 0.0;
   dart->init_speed = init_speed;
   dart->pptr = NULL;
   vec_scale( bmptr->orientation, init_speed, velocity );
   missile_util_comm_release_sub_munition( bmptr, MSL_TYPE_BALLISTIC,
                                         sub_mun, SUB_MUN_CANISTER,
                                         zero_vector, velocity );
#if DEBUG
   printf( "InitSpeed %1.21f Dist %1.21f\n", init_speed, dart->distance );
#endif
         **************
 * ROUTINE: missile_flechette_fly
               bmptr - Pointer to a _BALLISTIC_MISSILE_
 * PARAMETERS:
                       structure that's ammo-type is Flechette *
                       i.e. it releases sub-munitions of type
                       _munition_US_Flechette_60_.
               sub_mun - Pointer to sub-munition structure
                       associated with _bmptr_.
               veh_list - Vehicle list ID.
 * RETURNS:
               none.
               Simulates the flying of munition-type
 * PURPOSE:
               _munition_US_Flechette_60_.
               ~1200 2" lead darts are released and fly a
               cylindrical pattern 35 m in diameter ...
               Hence, we simulate the flechettes with ONE
               dart flown down the center of the cylinder
               and give it a 17.5 m proximity fuze. If the
                proximity fuze detonates, we impact the
               recipient vehicle and continue the lone dart's
                flyout to a distance of 750 m. At this point,
                the flechette rounds have lost the momentum
                and fall to the ground -- the rocket is
                terminated.
             ****************
int missile_flechette_fly( bmptr, sub_mun, veh_list )
BALLISTIC_MISSILE *bmptr;
```

```
BALLISTIC_SUB_MUN *sub_mun;
int veh_list;
    BALLISTIC_CANISTER *dart;
                        velocity;
    VECTOR
    dart = &(sub_mun->dart);
 * SPEED */
    bmptr->speed =
        missile_util_eval_poly( FLECHETTE_SPEED_DEG, flechette_speed_coef,
                                dart->distance ) + dart->init_speed;
 * DISTANCE */
    dart->distance += bmptr->speed;
    if( dart->distance >= sub_flech_char[2] )
        return( FALSE );
 * VELOCITY */
    vec_scale( bmptr->orientation[Y], bmptr->speed, velocity );
 * POSITION */
    vec_add( bmptr->location, velocity, bmptr->location );
 * PROX_FUZE */>
    if( missile_fuze_all_prox( bmptr,
                               MSL_TYPE_BALLISTIC, PROX_FUZE_ON_ALL_VEH,
                               &(null_VehicleID), &(dart->pptr),
                               veh_list, INVEST_DIST_SQ, FUZE_DIST_SQ ) )
        do
/* DETONATION ? */
            if ( missile_util_comm_check_sub_mun( bmptr, MSL_TYPE_BALLISTIC,
                                                  sub_mun, SUB_MUN_CANISTER ))
               missile_util_comm_release_sub_munition( bmptr,
                                                         MSL_TYPE_BALLISTIC,
                                                         sub_mun,
                                                         SUB_MUN_CANISTER,
                                                         zero_vector,
                                                         velocity );
         } while( dart->pptr != NULL &&
                 missile_fuze_detonate_prox( bmptr, MSL_TYPE_BALLISTIC,
                                             &(dart->pptr), FUZE_DIST_SQ, 0 ));
    return( TRUE );
 }
```

# Appendix P - Source code listing for sub\_m73.c.

The following appendix contains the source code listing for sub\_m73.c for convenience in document maintenance and understanding of the CSU.

```
/* SHeader: /a3/adst-
cm/RWA/AIRNET/simnet/vehicle/libsrc/libmissile/RCS/sub_m73.c,v 1.4 1993/01/28
23:27:09 cm-adst Exp $ */
* $Log: sub_m73.c,v $
* Revision 1.4 1993/01/28 23:27:09 cm-adst
 * P.DesMeules's changes for spcr 31
* Revision 1.3 1993/01/06 21:19:51 cm-adst
 * R.Branson's changes for the weapons model.
 * Revision 1.1 1992/09/30 16:39:52 cm-adst
 * Initial Version
*/
static char RCS_ID[] = "$Header: /a3/adst-
cm/RWA/AIRNET/simnet/vehicle/libsrc/libmissile/RCS/sub_m73.c,v 1.4 1993/01/28
23:27:09 cm-adst Exp $";
/*****************************
*, Revisions:
                                               SP/CR Number
                          Author Title
               Date
     Version
    1.2 10/23/92 R. Branson Data File Initiali-
                          zation
    1.3 10/30/92 R. Branson Added pathname to data
                          directory
    1.5 01/19/93 P.Desmeules Increased the size of the 31
                        fgets to make sure the
                        whole line is read in.
     ***********************
   *****************
      SP/CR No. Description of Modification
               Hard coded defines changed to array elements.
               Characteristics/parameter data array added.
               Added file reads for sub_m73 characteristics/
                 parameters.
               Added "/simnet/data/" to each data file pathname.
   ******************************
/*********************
 * FILE: sub_m73.c
* AUTHOR: Kris Bartol
 * MAINTAINER: Kris Bartol
```

```
This file contains routines which simulates
* PURPOSE:
              the behavior of sub-munitions of type
              munition_US_M73.
              10/06/90 kris
* HISTORY:
* Copyright (c) 1989 BBN Systems and Technologies, Inc.
* All rights reserved.
****************
#include "stdio.h"
#include "math.h"
#include "sim_types.h"
#include "sim_dfns.h"
#include "basic.h"
#include "mun_type.h"
#include "libmath.h"
#include "libmap.h"
#include "libmatrix.h"
#include "libmiss_dfn.h"
#include "libmiss_loc.h"
#include "rkt_hydra.h"
* Debug macro
*/
#ifdef FILEDBG
#define P(a)
#else
#define P(a)
#endif
                            /* debugging is ON */
#define DEBUG 0
 * Sub M73 characteristic parameters initialized to default values.
static REAL sub_m73_char[3] =
                     /* 75% of gravity - 75% * 9.8m/sec^^2/225 ticks^^2*/
  0.03266667,
                     /* bomblettes fall w/ +/- 8.8 deg angular displ */
  M73_FOOT_ANGLE_X,
                     /* bomblettes fall w/ +/- 12.35 deg angular displ */
  M73_FOOT_ANGLE_Y
};
static REAL zero_velocity[3] = { 0.0, 0.0, 0.0 };
static void missile_m73_get_impact ();
/**********************
 * ROUTINE: missile_m73_init
 * PARAMETERS: bmptr - Pointer to a _BALLISTIC_MISSILE_
```

```
structure that's ammo-type is MPSM
                       i.e. it releases sub-munitions of type
                       _munition_US_M73_.
               sub_mun - Pointer to sub-munition structure
                       associated with _bmptr_.
               speed - Terminal speed of Rocket at detonation. *
               none
 * RETURNS:
               Initialize rocket's _bmptr_ to behave according *
 * PURPOSE:
               sub-munitions type of _munition_US_M73_.
         ************
void missile_m73_init( bmptr, sub_mun, speed )
BALLISTIC_MISSILE *bmptr;
BALLISTIC_SUB_MUN *sub_mun;
                 speed;
REAL
{
    VECTOR impact_pt;
   VECTOR displacement;
      int
          i;
      float data_tmp;
     char descript[80];
FILE *fp;
      P(printf("$$$$ M73 file data $$$$\n"););
/* DEFAULT CHARACTERISTICS DATA FOR sub_m73.c READ FROM FILE
                                                                       */
      fp = fopen("/simnet/data/sub_m73.d","r");
      if(fp==NULL){
            fprintf(stderr, "Cannot open /simnet/data/sub_m73.d\n");
            exit();
      }
      rewind(fp);
            Read array data */
      /*
      i=0;
      while(fscanf(fp, "%f", &data_tmp) != EOF){
            sub_m73_char[i] = data_tmp;
            fgets(descript, 80, fp);
            P(printf("sub_m73_char(%3d) is%11.3f %s", i, sub_m73_char[i],
                        descript););
            ++i;
      }
      fclose(fp);
    END DEFAULT CHARACTERISTICS DATA FOR sub_m73.c READ FROM FILE */
    bmptr->time = 0;
    sub_mun->impact.timer = 0;
                                        /* distance rocket travelled last
    sub_mun->impact.distance = speed;
                                            frame, i.e. before detonation */
/*
```

```
* get point under sub-munition release point
    impact_pt[X] = bmptr->location[X];
    impact_pt[Y] = bmptr->location[Y] - 10;
    impact_pt[Z] = 10.0;
   missile_util_comm_release_sub_munition( bmptr, MSL_TYPE_BALLISTIC,
                                          sub_mun, SUB_MUN_IMPACT,
                                          impact_pt, zero_velocity );
}
      ****************
 * ROUTINE: missile_m73_drop
                 bmptr - Pointer to a _BALLISTIC_MISSILE_
  PARAMETERS:
                       structure that's ammo-type is MPSM
                       i.e. it releases sub-munitions of type
                       _munition_US_M73_.
               sub_mun - Pointer to sub-munition structure
                       associated with _bmptr_.
               TRUE if time of drop has been long enough to
  RETURNS:
               cause sub-munitions to hit the ground.
               FALSE otherwise.
               Simulation of the dropping of munition-type
  PURPOSE:
                _munition_US_M73_ rounds.
static int traj_up = TRUE; /* TRUE: vector UP -- FALSE: vector down */
int missile_m73_drop( bmptr, sub_mun )
BALLISTIC_MISSILE *bmptr;
BALLISTIC_SUB_MUN *sub_mun;
    BALLISTIC_IMPACT *impact;
    VECTOR impact_pt;
    impact = &(sub_mun->impact);
    if( impact->timer == 0 )
        if( missile_util_comm_check_sub_mun( bmptr, MSL_TYPE_BALLISTIC,
                                           sub_mun, SUB_MUN_IMPACT ))
        {
            if( impact->distance > 0.0 )
                impact->timer = (int)
                    ((8 * scaled_rand()) + 1.0 +
                     (sqrt((1.9 * impact->distance) / sub_m73_char[0])));
            else
                impact->timer = -1;
#if DEBUG
            printf( "Height %1.4lf Time %d\n",
                   impact->distance, impact->timer);
#endif
        else
            impact_pt[X] = bmptr->location[X];
```

```
impact_pt[Y] = bmptr->location[Y] - 10;
           if( traj_up )
               impact_pt[Z] = bmptr->location[Z] + impact->distance;
           else
               impact_pt[Z] = 10;
           traj_up = ( ! traj_up );
           missile_util_comm_release_sub_munition( bmptr, MSL_TYPE_BALLISTIC,
                                                  sub_mun, SUB_MUN_IMPACT,
                                                  impact_pt, zero_velocity );
       return ( FALSE );
   }
   else
       if( bmptr->time < impact->timer )
                                               /* wait until sub_mun's */
                                               /* hit the ground....
                                               /* incr time counter
           bmptr->time += 1;
           return( FALSE );
       }
                                                    ie. time == timer */
       else
           if( impact->timer > 0 )
               missile_m73_get_impact( bmptr->location, impact_pt,
                                      bmptr->launcher_C_world,
                                      impact->distance );
               missile_util_comm_release_sub_munition
                    ( bmptr, MSL_TYPE_BALLISTIC, sub_mun,
                    SUB_MUN_IMPACT, impact_pt, zero_velocity );
            }
/* reset time counter */
           bmptr->time = 0;
            return ( TRUE );
       }
    }
}
  *************
 * ROUTINE: missile_m73_impact
                 bmptr - Pointer to a _BALLISTIC_MISSILE_
 * PARAMETERS:
                        structure that's ammo-type is MPSM
                        i.e. it releases sub-munitions of type
                        _munition_US_M73_.
                sub_mun - Pointer to sub-munition structure
                        associated with _bmptr_.
                FALSE if all m73 have impacted the ground.
 * RETURNS:
                Simulation of _munition_US_M73_ impacts.
  PURPOSE:
int missile_m73_impact( bmptr, sub_mun )
BALLISTIC_MISSILE *bmptr;
BALLISTIC_SUB_MUN *sub_mun;
    BALLISTIC_IMPACT
                        *impact;
```

```
impact_pt;
   VECTOR
   impact = &(sub_mun->impact);
   if( impact->timer < 0 )</pre>
#if DEBUG
        printf( "ignore under ground detonation\n", bmptr->missile_id );
#endif
       return( FALSE );
   if( bmptr->time < 1 )</pre>
        impact->delay = 0;
                                       /* 0 - 0.250 sec delay */
   else
        impact->delay = (int)(250 * scaled_rand());
   bmptr->time += 1;
    if( missile_util_comm_check_sub_mun( bmptr, MSL_TYPE_BALLISTIC,
                                         sub_mun, SUB_MUN_IMPACT ))
    {
 * send _impact_ to util_ball & to world
        missile_util_comm_impact_ball_sub_munition( bmptr, impact );
        impact->quantity -= 1;
   get NEXT M73 _impact_location_
                                         OR stop
        if( impact->quantity > 0 )
        {
            missile_m73_get_impact( bmptr->location, impact_pt,
                                    bmptr->launcher_C_world,
                                    impact->distance );
            missile_util_comm_release_sub_munition( bmptr, MSL_TYPE_BALLISTIC,
                                                    sub mun, SUB_MUN_IMPACT,
                                                    impact_pt, zero_velocity );
            return ( TRUE );
        }
        else
            return ( FALSE );
    }
                /* Didn't get an impact */
    else
    {
        missile_m73_get_impact( bmptr->location, impact_pt,
                                bmptr->launcher_C_world,
                                impact->distance );
        missile_util_comm_release_sub_munition( bmptr, MSL_TYPE_BALLISTIC,
                                                sub_mun, SUB_MUN_IMPACT,
                                                impact_pt, zero_velocity );
                                              /* time's up */
        if( bmptr->time > impact->timer )
            printf( "M73_SIMUL timed-out: %d non-impacts\n",
                   impact->quantity );
            return( FALSE );
        return( TRUE ); /* keep trying */
```

```
}
static void missile_m73_get_impact( release_pt, impact_pt, mCw, height )
VECTOR release_pt;
VECTOR impact_pt;
T_MAT_PTR mCw;
REAL height;
                            /* Offset Vector in World Coords
    VECTOR detonation;
                                   of detonation point */
    REAL x, y;
    x = height * sin(deg_to_rad( sub_m73_char[1] * (0.50 - scaled_rand())));
    y = height * sin(deg_to_rad( sub_m73_char[2] * (0.50 - scaled_rand())));
    detonation[X] = x * mCw[0][0] - y * mCw[0][1];
    detonation[Y] = y * mCw[0][0] + x * mCw[0][1];
    detonation[Z] = - height;
 * Stretch _detonation_ vector to ensure intersection with ground/vehicle
    vec_scale( detonation, 1.5, detonation );
 * add to _release_pt_ to get location of _impact_ in World Coords
    vec_add( release_pt, detonation, impact_pt );
}
```